View mappings for query languages

Thesis

D 57032/85

UNRESTRICTED

VIEW MAPPINGS FOR QUERY LANGUAGES

Kam Chooi Wong

The Open University

A thesis submitted for the degree of

Doctor of Philosophy in the Open University

October 1984

Date of submission: October 1984

Date of award: 14 December 1984

DECLARATION

The work in this thesis is original unless otherwise stated.

I also declare that the work described in this thesis has not been submitted for any other degree.

ABSTRACT

The problems of current use of query languages are looked at. One chief drawback is the undesirable requirement for end user familiarity with and knowledge of the underlying database structures, in order to retrieve data effectively. The approach adopted towards resolving this is by means of high-level view support, using unit view structures called perceived records. A prime concern of this thesis then, is the study of perceived record mappings from the database.

A set of criteria for categorising and analysing the features of database mappings for end-user views is first developed. In addition, a classification of data structure transformations and data item transformations is also presented. The framework is general and is independent of a specific data model or database management system. Its usefulness is demonstrated by its application to the analysis of view transformations from recursive database structures to high-level, unit view structures. In addition, it serves as a basis for evaluating and comparing the mapping facilities in existing systems.

Possible ways of specifying a suitable data model for the perceived record view concept are described. Following on, two general mapping techniques are discussed. This leads to a proposal for a mapping mechanism that supports the flexible derivation of complex perceived record views that can differ considerably from the source structures. The mechanism uses an intermediary canonical transform model. Description of how the transform model mechansim can be used in practical systems to derive perceived record views, is also presented.

The feasibility of the ideas proposed are tested out by implementing an interactive software system for defining perceived record views. For this, a mapping definition language for perceived record derivation is first designed. The control system sets up the structures of the mapping definition language and prompts the End-User-Administrator to define and specify the mappings for a perceived record. Appraisals of both the proposed mapping mechanism and implementation are discussed. Examples of use of the interface system are included. The limitations of the implementation are pinpointed with suggestions for further improvements. Practical applications of the work and evaluation of the approach in the light of other existing approaches, are also discussed.

ACKNOWLEDGEMENTS

I wish to thank my two supervisors Mr. Mike Newton and Dr. Richard Maddison for all their help and guidance during the three years of research.

I also wish to thank all other staff in the Computing Discipline and in particular Hugh Robinson, for his valuable advice and interest.

Finally, I am grateful for the financial support of a studentship given by the Open University.

CONTENTS

TABLE AND FIGURES

Chapter 1 INTRODUCTION

1.1 Problem background

A structured database is a model of information about some real-world system, expressed as a collection of data values. This model is often massive and complex. A database system is an information resource tool that is designed to handle the complicated organisation and storage of data for the purpose of data accessing and sharing by many users.

The organisation of data refers to how data is grouped together in aggregates which relate to each other. It is specified by a set of rules defined as a logical data model. The storage of data requires data to be structured according to physical implementation considerations. An important criterion of data accessibility is user insulation of the details of, and changes if any, made to the storage and logical organisation of data. The latter type of insulation from changes is often termed as data independence.

A database architecture defines a framework for analysing the complexities of a database system in terms of interrelated components. One widely accepted useful reference architecture is the ANSI/SPARC [ANSI 75] model. This proposes three general levels of data structures: internal, external and conceptual. The internal level represents data storage descriptions; the external level features individual user's view of data, and finally, the conceptual level, lying between the internal and external levels, represents the description of the entire database. Each level is described by a **schema**; this is defined to be a collection of data types and representations.

Database users who access stored data, generally fall into two broad classes characterised by the level of expertise they have in manipulating the data. Sophisticated users are trained in the use of computer systems; these users comprise largely of application programmers.

End users are non-data-processing professionals whose job requirements will necessitate the use of a database system. Their needs are commonly data retrievals to satisfy some queries, although modifications to data may be performed as well. These users may be further categorised into skilled ones and casual ones. The former consists of:

- o ad-hoc users who very often use a structured query language. A query language is a high level language designed primarily for non-programming users chiefly to retrieve data from databases or files. Such users may include engineers, salesman, doctors or scientists.
- o transaction-oriented users whose requests are repetitive in nature and may invoke a tailored application program to obtain or modify data. These will typically be clerical personnel in an office or airline-reservation system.

The second category of casual, non-trained end users are not bound by any work requirements to access a database and hence infrequently do so. They may use a natural query language i.e. expressed in a natural language such as English.

User views in general, are just some portions of data in the database. The term 'view' used here is informal. A number of authors though, have used the term 'user-view' synonymously with the term 'external schema' [Zaniolo 79a, Clemons 79a]. We felt that such an equation is too loose; more often than not, the external schema in practice describes the mapping information as well, as Date [77] rightly pointed out. (The topic of mapping will be discussed later in the chapter).

In this thesis, we define a **view** to be a meaningful collection of data values that the user is interested in. The representation of these data values usually conform to a data model. View descriptions, such as information about its named contents and data structure, are held in the external schema. A view mechanism is an important means of suppressing irrelevant data in the database and hence abstracting only that for particular user's needs.

Significantly, the containment of only relevant data in views simplify access by users. It is this principal use of views as an access facility to users, precisely the ad-hoc users, that this thesis aims to concentrate on. It is expected that views in general for such end users will be predefined by an End-User-Administrator(EUA, c.f. DBA).

Other uses of views include the provision of data independence and its contribution as an authorization mechanism. The latter is achieved by allowing only users with the given privileges to access certain portions of data. This protects data from undesirable or wrongful manipulations and thus helps to maintain the integrity and security of the database.

For a long time, the database user community was mainly dominated by the application programmer. Not surprisingly then, many user views are designed to interface to COBOL or PL/1 programs. Probably the earliest form of a view mechanism is the subschema facility first proposed by the Codasyl DBTG in 1969 [Codasyl 69]. These subschemas views which interface to COBOL or PL/1 application programs, are limited to mere subsets of the database; users (application programmers) have to 'navigate' in order to obtain the data.

The rigid coupling to the underlying database structures has two consequences in that:

o it forces the conformation of user views to that of the database model (an inadequacy recognised in Nijssen [76], Pelagatti et al[78]).
o it does not allow for user's requirements and ease-of-use in a real-world application environment.

The first limitation generated much research efforts to provide multiple external schema views in terms of different data model representations of the database schema; a large number are devoted to deriving external schemas based on the hierarchical and/or relational data model from Codasyl network systems (See Section 2.2.1). These views consist of loose structures which are necessarily manipulated by the associated data manipulation languages (DMLs) pertaining to the data model.

Recognising the latter point made above, another line of research highlighted the need to orientate view structures to user's applications. A chief contributor was Clemons [78,79a] who challenged the limitations of the Codasyl subschema for programmer support and its lack of data independence. It became increasingly apparent that the way to extend the usability and usefulness of database systems is by abstracting views away from the system's constraints, to match user's requirements [Anderson and Dale 77].

In 1981, the BCS Query Language group published a report [BCS QL 81] which underlined the shortcomings of query languages for end users. The main thrust of the report is aimed at a query language system that supports a uniform approach towards data access for end users, irrespective of how data is stored and structured in different databases (or files).

A fundamental problem identified by the report is the close linking of query views to the database structure. A number of existing database query languages are based on the relational data model. Examples are SQL, QUEL and ISBL. These allow users to retrieve data in terms of database relations or in terms of defined view relations. Most other query languages which operate on views of network or hierarchical database are similarly constrained by views that are often coupled to varying degrees to the database model (e.g. Univac's QLP). Consequently, query language users (this term is used synonymously with end users who use a query language) are faced with the task of stating explicitly **how** pieces of data are to be derived or connected together in order to use the system effectively to retrieve the required data values. This inevitably requires some familiarity with database structures.

Certainly it was on the basis of this requirement that attempts were initiated to ease the problem by allowing users to pose queries in terms of named data items only. One early implementation is the APPLE system [Carlson and Kaplan 76]. This is followed by more recent universal relation systems. Alternative proposals, running along very similar argument for ease-of-use, are natural query language systems. Examples include Codd's Rendezvous [Codd 74], ROBOT [Harris 78], TORUS [Mylopoulous et al 76], EUFID [Kameny 78] and that by Waltz [78]. Experiments comparing natural and structured query languages, have shown though, that an essential ingredient in effective query language usage seems to be the structuring of data that users can perceive and understand [Shneiderman 78]. It is on this basis, that we shall remain interested only in structured query views in the rest of this thesis.

A second finding by the report highlighted on a wider perspective, the non-uniform approach to using query languages in general, to extract data. Each query language is based on a different form of view structure or database structure. If end users wish to access data from multiple database systems that are different (such as from a network, relational and hierarchical system), they must learn to cope with varying types of data-model dependent view constraints; there is no one common query language.

This thesis thus specifically researches into the derivation of views that overcome the above limitations. In particular, a view model is proposed that can provide the basis for specifying simple, DBMS-independent query commands to retrieve data. Views defined by the model are called perceived records; the terminology is first credited to the BCS Query Language Group. The work involves a study of end-user view requirements and methods to derive them. The term 'end users' for the rest of the discussions will now be taken to refer strictly to those who use a structured query language.

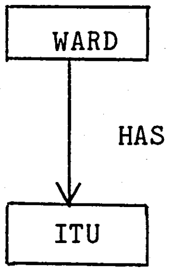
1.2 Characteristics of a high-level, user-oriented view

The characteristics can be regarded from the point of contributing towards, firstly, the functionality and secondly, the user-perception of the view.

1.2.1 Functional requirements

The chief purpose of a query view is to facilitate direct data access. This requires that where necessary, raw data from the database can be processed in a 'ready-to-use' form so that users do not have to specify additional procedures in order to retrieve the desired data values. The following example based on a HOSPITAL database illustrates the point.

The database describes about wards, patients, doctors, nurses and their related information in the hospital. Suppose some information about a ward, modelled in the database is thus: each WARD(entity) has INTENSIVE CARE UNIT(entity, abbreviated to ITU) as shown graphically below.



The corresponding attributes of the two entities are:
Ward(W-no, W-name, Matron-name, No-of-beds)
ITU(U-no, No-of-beds)

In order to answer a query of the general form 'How many beds are there altogether in a ward?', the end user should not have to understand:

o that additional information i.e. no-of-beds are held in another entity i.e. ITU as well.
o how to access this information.
o how to compute the desired data by adding no-of-beds from ITU to that in WARD.

In other words, a view consisting of information about a Ward(note: not Ward and Intensive Care Unit) should be available to the end user with data items, W-no, W-name, Total-no-of-beds in the view. The query then becomes a direct retrieval request.

The example here sums up two aspects central to the functional requirements:-

o the provision of data values in the desired form.
  Often enough, end users are not just interested in getting raw data values from the database, but wish to extract information in the form of computed values, such as a summarized statistical value, for a given view application.

o the achievement of user independence from logical database structures. A solution to this is to capture all the data values representing the user's information needs in a unit data structure. A unit view structure therefore defines a 'complete' semantic unit of information to the end user; specifications of connections related to the underlying structural constraints are hidden away. The end user is thus relieved of the task of 'navigating' between disjointed data structures. Complete is used in the sense of satisfying some given information requirement about a user-perceived real-world object. This refers to the primary entity that the end user wants information related to, and on which the whole meaning of the view focuses. The view may contain information drawn from other entities (as in the above example, from ITU) but the end user should not have to perceive or understand as so.

At this juncture, we introduce the concept of a **base entity** (or otherwise referred to as root entity) of a view, that is mappable from the database which represents the user-perceived real-world object. For instance, the base entity of the above view example is the database entity WARD.

A second important consideration is how the end user can use such a view effectively, given that we now have a view that contains the desirable functional characteristics.

1.2.2 User perception and semantics

One factor which significantly affects the understanding of the meaning, and hence the use of the view, is the end user's perception of it. This in turn depends on the way that data values are structured. A data structure embodies certain semantics of the data within it. The chosen structural representation should reflect as closely as possible the real-world situation. Often, in many practical applications, data is perceived in a hierarchic form. For instance, in the case of ward with many patients, one data occurrence of WARD has repeated occurrences of PATIENTs associated to it. A comprehensive data representation for such kinds of views is a hierarchic one.

In accordance then, a second view requirement is the capability to structure and represent repeating data values where necessary. Furthermore, in order to aid fuller perception of the view's semantics, the need exists that meaningfully associated data values can be grouped into an identifiable object in the view.

The points considered in this and the previous section can now be best concluded with a more complete example. This again uses the HOSPITAL scenario.

Consider the following view PATIENT which describes the user-perceived object of interest; the corresponding base entity is the database entity PATIENT but the view also includes other related information of interest.

PATIENT(P-no, P-name, Appmt(Appmt-date, Doctor(D-name, D-phone-no)),
Test (Type-of-test, Result))

All data values are represented as data items which are not underlined; these are P-no, P-name, Appmt-date, D-name, D-phone-no, Type-of-test and Result. Note that the end user will not consider the last five data items (from Appmt-date to Result) as belonging to independent and separate entities. He or she will regard them as the Patient's Appmt-date, Patient's Doctor(name) and so forth, i.e. information pertaining to the base entity of PATIENT view.

The underlined names of Appmt, Doctor and Test serve as aggregate names for the purpose of enhancing the semantics; they do not contain any data values. The meaning of the view is that each patient can have one or more appointments, and one or more tests. Each appointment is with one doctor only. In short, the data values of Appmt-date, D-name, D-phone, Type-of-test and Result can repeat.

An occurrence of the view PATIENT is as follows:

P-no	P-name	Apmmt-date	D-name	D-phone-no	Type-of-test	Result
123	B.Smith	10-1-83	T.B.Connors	653120	Eye	positive
		12-1-83	J.S.Bank	653482	Skin	negative
		20-4-83	R.W.Ling	653125		

1.3 Mapping

Intrinsic to a view mechanism is a mapping. This is defined to be a mechanism which specifies how one set of data object descriptions and constraints at one level corresponds to a set of data object descriptions at another level. A data object here refers to a data type or data value. The two levels of concern here are the external and conceptual level.

The correspondence can be classified into two types: structural and operational. In this context, structural correspondence is the translation of both the inherent constraints of a set of structures (as defined by its logical data model) and the constraints defining the data values, from the conceptual level to those at the external level. An inherent element of structural correspondence are the kinds of changes in the semantic information that may be obtained with any differences in the data structures.

Operational correspondence is the translation of operations specified at the external level to those that can be performed at the conceptual level. In both cases, the equivalence or correct translation, is a crucial requirement of a mapping. Translation of retrieval operations, in general, are fairly simple. Those of update operations, on the other hand, are tricky because they can affect the global data in the database. Any modifications must be carefully propagated in order that the database maintains its integrity.

In this thesis, we specifically consider structural correspondence from the conceptual to the external schema, in the context of retrieval operations only. Operational modifications to data in the external schema will not be translated to the conceptual level, the chief reason being that end users are more commonly interested in data retrievals. Other works which consider the mappings of update operations can be referred in Clemons [76b], Klug [78] and Dayal and Bernstein [78].

The conceptual schema objects used for our investigation are entities, attributes and relationships (e.g. the EAR model as described in OU [80], or that on which Chen's Entity-Relationship model [Chen 76] is based on). Internal schema storage descriptions will not be relevant in the study.

1.4 Organisation of thesis

The thesis can be regarded as having two main parts. The first part deals with the development of a generalised framework to categorise mapping concepts and features and the subsequent application of these findings to a. analyse view transformations from recursive database structures and to b. evaluate and compare mapping facilities of practical systems. The second part concerns the specific research into perceived record mapping -- i. a definition of a view model for perceived record, ii. a proposal for a mapping mechanism and a design and iii. implementation of an interface facility for perceived record definition. The corresponding chapters are as follows :

Chapter 2 reviews past research into mappings of different kinds of views and their relevance to that in this thesis. For this, a framework for categorising the various classes of works is used.

Chapter 3 formulates a framework of criteria for a mapping facility. This consists of five distinct concepts that form inter-related features of a mapping facility. A classification of data transformations, sub-divided into data structure transformation and data item transformation, is also presented. The features and requirements of two other chief aspects of a mapping facility are also discussed. These aspects are : i. the corresponding mapping definition language used to define view mappings, and ii. the process of how the definitions and mappings of views is carried out.

Chapter 4 describes the analysis of mapping unit view structures from recursive database structures based on the framework developed in Chapter 2. The study represents a new way of looking at such transformations which emphasises the abstraction and preservation of the semantics in the underlying database structures, and the representation of data in views.

Chapter 5 uses the mapping framework to evaluate the mapping capabilities of System R and those proposed by the Codasyl EUFC. In this way, the framework provides a common basis for comparing the available facilities.

Chapter 6 concerns the specification of the perceived record view model. Three different methods of modelling the characteristics of the perceived record view are described and evaluated.

Chapter 7 develops a mechanism for mapping perceived records from the database. Existing techniques of mapping are first surveyed. The subsequent approach adopted is an intermediary one, in the form of a canonical transform model. Its applicability in a practical system, advantages and disadvantages are also discussed.

Chapter 8 describes the design and implementation of an interactive mapping interface which controls and prompts the EUA to define a perceived record and its mapping from database types. A mapping definition language for the purpose is presented. The use of the facility is demonstrated by example runs given in the appendices.

Chapter 9 concludes the thesis. A summary is given and directions for further extensions to the existing implementation, and practical applications of the research are suggested. Finally, an overall appraisal of the perceived record approach is briefly discussed in the context of general database applications and other existing approaches to resolving the problems of database querying.

Chapter 2 REVIEW OF RELEVANT RESEARCH

2.1 A characterisation

The wide spectrum of research in external-conceptual database mappings can be broadly categorized according to the degree of difference firstly, in the syntactic data structures and secondly, by changes in the semantic content of some pieces of data from the conceptual level. These two criteria could be said to be due first to Lisboa [79] who used them as the basis for classification of work in this area. His framework was subsequently adopted by Spaccapietra [80] in the treatment of mappings in heterogeneous distributed database systems. This specifically pinpointed the differences in the data model and semantic interpretation (SI) that can exist at the two levels. Semantic interpretation was simply defined as 'the interpretation by one of the semantics of a piece of reality' [Spaccapietra 80]. Four categories in all were identified:-

o Trivial Mapping - This establishes no change in data model representations. The external semantic interpretation (E.SI) is equivalent to or a subset of the conceptual semantic interpretation (C.SI), e.g. as exists with many commercial query products.

o Semantic Mapping - This again establishes no change in data model representations but the E.SI now differs from the C.SI. Relational query systems such as Ingres and System R provide good examples here.

o Model Mapping - This establishes a specific change in data model representations. The semantic interpretations remain unchanged. The multiple schema work by Zaniolo [79a] is representative of this.

o Total Mapping - This is a combination of semantic and model mapping; both the SI and data model representations differ at the two levels. An example is the high-level user interface implemented by Clemons [76a].

These mappings are illustrated in fig. 2.1.

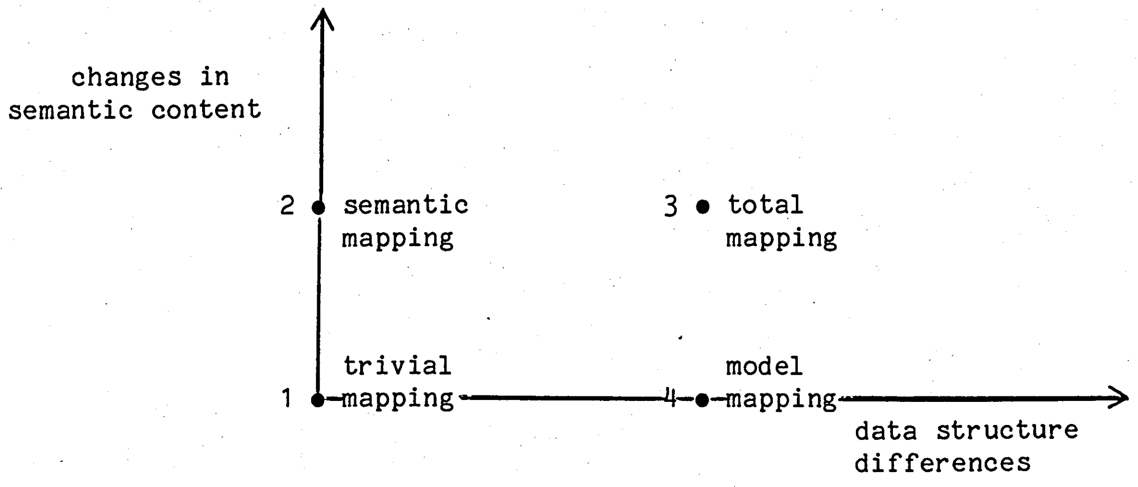


fig. 2.1 Categorization of External - Conceptual Mappings

In yet another paper, Anderson and Dale [77] defined three classes of mappings at a more general level i.e. not necessarily external-conceptual. The classification is in terms of data value changes or correspondences:

- o Value transformation - this is simply the mapping of one value set to another.
- o Schema directed transformation - the key factor here is that individual data values do not get transformed; the schema structure is manipulated or transformed.
- o Value directed transformation - this is recognised as the most important category; it establishes substantial changes in both the data structures of the schema and the data values, and hence the overall semantics that can be derived. (c.f. total mapping).

Although some useful concepts in terms of the power and capabilities required of a view mechanism are presented, their categorization is not very complete; it does not sufficiently cover the range of research in this area.

The remaining sections of this paper will use the top three realms (1, 2, 3) of the graph in fig 2.1 as focal points for referencing the works of other researchers. An attempt is made to assess the impact on, or perhaps the departure from the line of research pursued in this work, which falls into the realm of total mapping. The majority of contributions are from within a single database environment. The last section is specially devoted to examining work done in integrating distributed database systems, which in terms of mapping requirements bear direct relevance to the work in this thesis.

It is noted that trivial mappings provided by most commercial DBMS(s) will not be dealt here; these are mainly subschema facilities. An appraisal of query languages and the provision of view facilities in a list of Codasyl-based DBMS can be found in Tagg [83].

2.2 Contributions along the axis of data structure differences

2.2.1 Specific data model mapping

Much research has been generated in this direction over the past years. The mappings involved emphasise the structural correspondences of inherent constraints between the data models and the operational correspondence of the data manipulation commands.

A number of contributions in particular have been made with Codasyl-type network databases. Zaniolo [79a, 79b] designed algorithms based on key migration for mapping relational and hierarchical views over Codasyl-78 [Codasyl-78] schemas. Emphasis is laid on preserving the information contents of the underlying database in deriving general purpose views which can support both query and update operations. Goldman [79], on similar lines, developed a system to automatically translate Codasyl schemas to a relational schema. Translations of hierarchical and network database schemas to external relational schema views in order to support transactions (restricted to retrival operations) in a distributed database environment have been investigated by Vassiliou and Lochovsky [80]. They specifically considered the problems of mapping basic relational operators (SELECT, JOIN and PROJECT) in the queries to corresponding operations in the underlying database schemas. Another contribution which specifically looked at the mappings of the relational operators (JOIN and GROUP BY) on navigational network schemas is due to

Esslemont and Gray [82]. More recent work on the correspondence of Codasyl network schemas to relational schemas can be found in Fallahzadeh and Cousins [83], who identified the limitations of the other three pieces of research mentioned above. However, an important issue which all these works did not attempt to address is mapping repeating groups from network schemas. This thesis extends to consider transformations of repeating data structures into view structures.

Related research on the mapping from relational schemas to differing data model external schemas have been carried out by Kalinichenko [76] and Lien [81, 82]. The former concentrated on the network model while Lien [81] designed (2-steps) algorithms for deriving hierarchical schemas. His other contribution, (Lien [82]) specifically studied the problem of equivalence i.e. preserving the constraints in the mappings between relational and network acyclic (loop-free) schemas.

At a more general level, Klug [81] proposed a relational canonical mapping model that supports the derivation of hierarchical, network and relational external schema views. This, in essence, is the translation of the external data model and the relational model. Some other work [Date 75, Klug and Tschritzis 77] which concentrated specifically on the mapping of DML operations of the hierarchical, network and relational data model, proposed a single common language to support the different types of data structures being manipulated.

The works mentioned so far are concerned with the three conventional data models: hierarchical, network and relational. There are some other efforts which involve different data models. Pelagatti et al [78], for example, demonstrated the mapping of a binary relational model at the conceptual level to the n-ary relational model [Codd 70], using an algebraic specification. A semantic data model is the basis of the translation of a Codasyl database schema to a relational database schema in Biller's work [Biller 79].

The references described above are by no means exhaustive; it is only hoped that they can provide adequate illustrations of the contrasting emphasis in the other remaining sections of this chapter.

Finally, it is noted that this type of mapping is intended to support high level DML(s) i.e. data access via commercial application programs. Notably, the change in view representation aims at the adaptability to using a different data model based DML. There is no change in the basic interpretation of the database semantics; views are represented in disjointed structures (unlike in a unit view structure) and in general, no processing of data values occurs as part of the mapping process.

An interesting development in recent years has been to facilitate data access via high level programming languages using data abstraction software tools. The following description is intended to give a more complete picture of the current state-of-the-art in mapping techniques via different types of computer languages.

2.2.2. View Mappings via high-level programming languages

The general term of data abstraction refers to the containment of only essential data and the exclusion of irrelevant details, required for a certain purpose or certain user needs. Data abstraction is a fundamental concept in reducing the complexities of using computer systems. In database systems, it forms the underlying principle of view facilities. It is also the basis for developing database design tools [Smith and Smith 77a, 77b] and for designing semantic data models [Hammer and Mcleod 78, 81, Brodie 81].

In the area of programming languages, the emphasis of abstraction techniques has been to develop software tools for clearer and shorter specification of problems [Jones 80]. Research has shown that such tools can be used to directly define database views in a high level programming language environment [Weber 79, Wasserman 79a].

A number of high level languages have been designed with the aim of integrating database objects into the program without the need of the programmer explicitly mapping database structures to conform to the data types of the program. (This mapping is usually performed by coding calls to the database structures). This type of integrated languages [Lacroix and Pirotte 83] are largely based on the programming language PASCAL, for accessing a relational database. The list includes RIGEL [Rowe and Shoens 79], PLAIN [Wasserman 79b], PASCAL-R [Schmidt 77] and EXT-PASCAL [Claybrook 82, 83]. The use of abstract data types [Jones 80] in these languages has contributed to a means of defining database views which encapsulate, and are characterised by a distinct set of predefined operations that can be allowed in the views. This certainly adds a new dimension to the conventional type of database views that are described in an external schema, in that it is not just a way of restricting what kinds of data the user can access (and hence simplify the user's task), but also determines what types of operations can be performed on the data. It is therefore particularly useful in controlling and monitoring the consistency and integrity of the database in a shared update environment [Weber 79]. On a slightly different slant, the attractive property of encapsulating operations within a data type has prompted Lockemann et al [79] to apply abstract data types in the design of databases.

Another piece of work which adopted the concept of data abstraction based on the persistence of data objects in the programs, is due to Atkinson et al [83]. The programming language they proposed is PS-ALGOL (developed from S-ALGOL); the approach to defining database views and the objectives are in general, similar to those in the other languages mentioned above.

In a contrasting contribution, Leavenworth [81] uses a data abstraction language in conjunction with a module interconnection language to define views of the hierarchical database IMS [as described in Date 77]. Both languages are very like CLU [Liskov 77]. The data abstraction object (or view) is a two-dimensional form (which is due earlier to Housel and Shu [76]. Interestingly, a form can either be a 'flat table' (i.e. in the relational sense) or a hierarchical data structure (c.f. the characteristics of a view as described in Section 1.2).

2.3 Contributions along the axis of semantic changes

This can be divided into two lines of research directions. The first establishes different set of view semantics by shuffling and reorganising pieces of data in the database schema; individual data values are not operated on i.e. modified or transformed. The second involves actual transformations of data values which registers the changes in the meaning of the data. This is especially true of research-oriented relational query language systems. Notably in both cases, no change in data model representation is specified.

2.3.1. Restructuring of schemas

Extensive work has been carried out in the restructuring of hierarchical schemas [Mehl and Wang 74, Navathi and Fry 76, Dale and Dale 76, 77, Navathi 80]. Navathi and Fry identified three types of schema modification: renaming of the schema constructs, combining of schema constructs and relating of schema constructs (constructing new relationships between the source schemas). Each type of modification has corresponding effects on the associated occurrence structures, expressed by instance and value operations. (Note that the 'value operations' here are non-value modifying: these operations are delete value, copy value and create null value).

Dale and Dale presented useful algorithms for restructuring classes of schemas: subschemas, ancestor schemas and descendent schemas, from an original hierarchical schema. They demonstrated that these derived schemas are consistent in that queries defined on these schemas will be equivalently processed and executed on the original schemas.

The restructuring techniques used for hierarchical schemas are mainly operation-based. Swartwout [77] argued that such operation oriented specifications are not applicable to network schema restructuring. The approach put forward instead, was a descriptive specification of the desired target schema in terms of the access paths which relate the source data types.

In accessing the relevance of these research in terms of direct access functional requirements, the efforts represent one step towards provision of semantic information that can be derived to satisfy user needs. However, because the data model remains unaltered, any data retrieval will still require some knowledge of the linkages in the data structures. In addition, any data value conversion must be specified externally by the users themselves.

2.3.2 Modification of schemas by queries

Two of the most prominent contributions in this area are System-R [Astrahan et al 76] and Ingres [Stonebraker 75]. The concept of views predefined in an external schema by a database administrator (as exists in Codasyl database systems) is absent. Users define their own views directly in terms of underlying database relations, in a query language, by means of query modification algorithms. A view in these systems specifically refers to a derived relational table. Capabilities are provided which support a large range of data value transformations. The view mechanism of System R will be assessed in greater detail in Chapter 3.

Universal Relation Systems

A great deal of research and literature has been generated about the concept of the universal relation. The topic is still undergoing active research and there is still not yet a general acceptance of precise definitions of the many ideas associated with it. Indeed, as recently as 1982, Ullman [82] gave a taxonomy of the various assumptions referred to generally as the 'universal relation assumption' that appeared on numerous papers. He identified five different forms altogether. We do not propose to go through all five in detail; we shall only describe one which is fairly commonly encountered. This is known as the Universal Relation Scheme Assumption (URSA). The underlying assumption here is that sufficient renaming of attributes occurs such that there exists a unique relationship among any set of attributes, in order that 'all attributes are available freely for combination into relation schemes' [Fagin et al 80]. This assumption has important implications in the design of universal relations as a user interface.

For a long time, conventional relational query language systems (such as System R or Ingres) have been credited with freeing the user from the knowledge of physical access paths [Date 81]. However much criticism remains about the fact that an understanding of the logical associations between the database relations is still necessary in specifying a relational query [Maier 83a, Biskup and Bruggemann 83]. The universal relation is seen as a means to overcome this latter limitation. The user queries the database in terms of attributes only. It essentially appears as if the database were one single relation*; he or she is relieved from learning the intricacies of the database structures [Ullman 82, 83].

A number of query language systems have been developed based on the universal relation concept. An early study was made by Carlson and Kaplan [76] in which they proposed an attribute-based query language called APPLE (An Access Path Producing Language). Another system called q [Aho and Kernighan 80] is still under current development. Other recent systems include System/u [Korth and Ullman 80, Korth 81, Ullman 82], PIQUE [Maier et al 82, Rozenshtein 82, Stein 83] and Parafrase [Kuck et al 80, Kuck and Sagiv 82]. Kuck and Sagiv designed a universal relation interface over a network database. The PIQUE system uses a query language called PITS which is described in [Maier 83b].

All these systems rely on a capability to infer required linkage that exists between sets of attributes (the consequences of it not being correct need to be explained later). For this, it is assumed that 'among any set X of attributes, there is one basic connection that may serve as the default connection' [Ullman 83]. This basic connection for mapping is referred to as window [Maier 80] or connection [Maier et al 82].

Much scepticism has been directed at the suitability of the universal relation scheme as a user interface [Atzeni and Parker 82] and more generally, the underlying implications behind the assumptions [Kent 81]. The main objection to it as a user view can perhaps be best summed up as the sheer dependence on the database query system to deduce the 'correct answers' to users' queries; this it is argued, relies heavily on natural inference and intuitive assumption of what the user wants -- it is

* Some gross simplifications may have been implied. In this rather brief review, it is not possible to describe every detail; the reader is referred to the appropriate literature where necessary.

especially vulnerable where semantic ambiguities can arise with cyclic schemas. Ullman [82] however, recognised this fact and pointed out that the query language should allow the user the option to specify a connection in his or her query if so desires, but that the simpler connection (in the case of several) be used as the default if a specification to the contrary, is not stated. However, this begs the question (an expression of the author's own opinions) because the user will only be able to define exactly which connection to use if he or she is aware of what is in store in the underlying database.

It is perhaps appropriate at this point, to compare the approach put forward in this thesis and the universal relation approach towards the design of user views. Just as with universal relation schemes, the prime objective of a view in this research, is granting independence from details of underlying logical database structures to the user. The proposal here for achieving this is to encapsulate a set of distinct semantics within a set of data values (associated with the corresponding data items), which conform to a semantic view object. This, we believe, overcomes the types of conflicts inherent in the universal relation approach mentioned above. For each connection that exists between two sets of attributes, a semantic view can be defined where necessary and the differences made clear and explicit to the end user.

2.4. Specific User Interfaces

Contributions in this section are representative of total mapping. A significant implementation was by Clemons [76a, 76b] who designed a user-oriented interface for the data processing environment. The interface consists of hierarchically-structured data objects called Virtual Information Objects (VIO's) that represent a COBOL or PL/I record, mapped from a relational database; these VIO's can then be easily manipulated easily by a COBOL or PL/I application program. He identified three important aspects of a mapping description:

o Format information which describes how a VIO can be structured differently from the database relations; for this, a taxonomy of a hierarchical unit view structure is presented.

- o Access information which describes the access paths (relationships) that are to be used in deriving the source data items.
- o Data item information which allows the transformation of values of data items.

VIO's may be virtual, defined in the sense that they may be used in the description of other VIO's as if they were relations. A particularly interesting aspect of the work is the processing of recursive hierarchical data structures (the mappings of which are still not well explored) into amenable VIO structures. In subsequent works, he extended these ideas to propose an external schema facility to replace the limited subschema facilities for Codasyl-78-based systems [Clemons 79a, 79b].

A very relevant proposal for a user interface is the Codasyl End-User-Facilities Committee forms-oriented approach [Codasyl EUFC 83]. The interface is aimed at end users in an office environment, where views of a Codasyl database are structured and perceived as forms, which may be considered as normal office forms or documents. The proposed mapping technique is via elementary data items; a full evaluation of this proposal is dealt with in Chapter 4.

Earlier on, Dale and Lowenthal [77] discussed a very similar approach, based on forms advocated by the then known Codasyl End User Facility Task Group [Codasyl EUFTG 75]. In particular, they applied previous research in hierarchical schema restructuring [Dale and Dale 76, 77] firstly, to the mapping of hierarchical forms from a database schema which is restricted to a hierarchy, and secondly, to the definitions of forms on top of existing hierarchical forms.

All three studies have one thing in common: the proposal of a single hierarchical data object as the user view. This serves to reinforce the similar stand taken in this thesis; further reference pointing towards the same direction is that of Leavenworth [81] described in the preceding Section of 2.2.2.

Moving away from the conventional data models (and specifically the relational data model), McLeod [82] adopted a different, alternative approach to end-user interfaces and in particular, universal relation interfaces in granting user independence from logical database structures. He presented an interactive system which prompts and guides the naive end-user to formulate and identify the required data from the database by a stepwise refinement methodology. The interface is built on an underlying semantic database model called SDM (Semantic Data Model), developed by Hammer and McLeod [78, 81]. The construct which represents the user's perceived real-world object is a 'class' in the SDM. Although such a methodology may eliminate some of the semantic limitations of the universal relation approach or natural language approach, it is however not as appropriate for the class of end users we are interested in.

2.5 Contributions from integrated distributed database systems

The direct relevance of research in this area is the mapping of multimodel (heterogeneous) database schemas into a single common data model, on which users can express queries independently of any specific local distributed DBMS.

A very interesting software system designed for such integrated access is MULTIBASE [Smith et al 81]. The system supports a unified global schema which is expressed by the functional data model [Shipman 81]. Corresponding to the functional data model, the single unified query language provided for user access is DAPLEX which is proposed by Shipman [81]. The entire project is to span over three years and at the time of the report publication, three types of local DBMS schemata were claimed to be integrated into the global schema: the Codasyl model, the relational model and the (commercial) file model. The mappings involve translating the data structures and constructs of each of the three models to entity types and functions in the functional data model. The complete mapping from the local host schema (at the site of each specific DBMS) to the global schema is performed in two stages: (1) the translating of the local host schema to a local functional-model based schema, (2) the merging of the transformed schemas to the single global schema. A particular aspect is the integration of incompatible data into the global schema. Three types of data incompatibility that can occur are given as:

- Scale difference - one example is that climate may be assigned one of four values (cold, cool, warm, hot) in one database, or it may have a numeric value of average temperature in Fahrenheit in another database.
- Level of abstraction - an example described is the recording of an employee's average salary over the past five years in one database, while another database may record the 'salary history' instead.
- Inconsistency of data - the same piece of information occurring in several databases may have different values due to some errors.

The distributed context of the mappings has given the work a generality that has been missing from all others mentioned above -- these are either restricted to a particular data model or are DBMS-specific. Effectively, the transformations of local schemata to the global schema represents a model mapping whilst the resolution of data incompatibility is one form of semantic transformations. More precisely, it characterises a total mapping. Notwithstanding the judgement of whether a view model based on a unit data structure is better than a functional data model in terms of end-user usability, the MULTIBASE system does share in common many of the mapping requirements underlined in this thesis -- general-purpose, ease-of-use and support of a unified high level query language. Finally, it goes to show that view mappings carried out in this thesis can be usefully applied to a distributed database system.

Chapter 3 A FRAMEWORK FOR CATEGORIZING MAPPING FEATURES

3.1 A set of underlying concepts

There are five essential features of a mapping facility.

3.1.1 Data structure support

It has been noted in previous chapters that the structure of a view contributes significantly to the understanding and interpretation of some pieces of data. The facility to define a view structure suited for end users is thus a fundamental aspect of the mechanism. The categorization in Chapter 2 has shown that not all view facilities incorporate this capability; in many instances, the structure defaults mainly to the database schema (subschemas), or is just a list of data items (as with many commercial products). In the latter case, the view structure is often represented as the displayed format resulting from some query processing.

The view characteristics as described in Section 1.2 require the support of a single hierarchic data structure. The concept of a base entity of a view will then be represented by the root node of a tree hierarchy. Associated with a view structure is the corresponding view occurrence structure. This can be thought of as the (sub) set of data occurrences selectively combined as a whole unit from occurrences of the source data types. In specifying the hierarchic structure of the view, the external schema needs to contain the database types and the derivation rules for the proper construction of a view occurrence structure.

3.1.2 Naming

Naming is an essential aspect of a mapping mechanism where there is a need to reference database objects by perhaps more meaningful names, or to reference newly derived data objects in the view. The provision of names is a fundamental means of referencing data.

Meaningful names clearly contribute to the understanding of the semantics of the view. However, names cannot adequately serve as a proper explanation of the complete semantics of the view. Such an explanation can be appropriately provided as a data dictionary facility that stores the textual description about the meaning of the view and its data items.

The most basic form of (re) naming in a mapping is of atomic data type i.e. a data item type in the view or a database attribute type. Naming is an important aspect in the structuring of atomic data types into a higher level or aggregate data type. Such a data type within a view, is a collection of data items in the view which have a common association. The proper semantic naming of the aggregate structure brings out the meaning of this association. This aids the end user's comprehension of the view; it is especially true with complex views. Referring to the PATIENT view on page 8, three such aggregate names Appmt, Doctor and Test are used to present a clearer context of the view to the end user.

3.1.3 Format

This refers to the representation of data values of the view as displayed to the end user. It relates to both the individual data value representation and the overall spatial layout of all the data values for an occurrence of the view.

The representation of a data value may be numeric, for example, as integers or real numbers, or it may be character string. The term data type used elsewhere [Date 77] is synonymous with data item format. The conceptual schema, in principal, is not concerned with data representation. The representation of individual values is therefore an essential aspect of a mapping. In practice however, (logical) schemas do incorporate the representational format, expressed in terms of domain values. Consequently, the specification of a data item format is an optional part of a mapping. It is only required when the desired display format differs from its schema type format, or when a standard default transformation (as exist in most programming languages) is not appropriate.

Section 3.1 earlier advocated that the overall representation of data in a view be supported by a hierarchic data structure. This basic structure may be mapped to a display layout designed specifically to enhance user's perception. Such a format is a presentation of the spatial positions of data values relative to each other, (most likely laid out on a screen), with the underlying hierarchical framework still retained. The use of a form layout, for example, is claimed to be most suitable for the office environment c.f. the Codasyl EUFC proposals. These proposals contain a Geometry Section which is used to specify the mapping of hierarchically-structured derived values to the geometric positions in the user-perceived form. One can even go further to consider graphical facilities mapped on top of the hierarchic structure. However, although such facilities have their relevance towards the perception of data, they are not a fundamental feature. For our intended view model aimed at general query purposes and direct retrievals, the simple linearised view representation as illustrated on page 8 is adequate.

3.1.4 Selection

There are two forms of selection. Selection of data types and the exclusion of irrelevant ones, is a basic means of limiting data contained in a view. Type selection is a facility fundamental to many aspects of database systems. It is, for instance, an integral part of query language (sub) systems.

Selection of data occurrences, on the other hand, is not as commonly supported as a means of view definition. The inclusion of this capability is important because it is a way of conceptualising new classes of semantics that can be logically derived from the underlying database semantics, which has not been defined before [Hammer and McLeod 81]. Suppose in the HOSPITAL database example (Section 1.2), the database entity APPOINTMENT has APPMT-TIME as one of its attribute. The attribute APPMT-TIME does not necessarily have to appear as a data item in the view

LATE-APPMTS. The inherent meaning of 'lateness' derived from some property of APPMT-TIME is now implicitly carried in the view. Occurrence selection is therefore a way of creating new view objects which can encapsulate some common property.

3.1.5 Identification of data values for a view occurrence structure

One of the primary objectives of a view mechanism (as underlined in this work) is to free the end user from specifying database connections when using a view. In order to achieve this, the mapping facility must support the selection of data values for an occurrence corresponding to the defined data types that comprise a view structure, based on correct unambiguous links between the data types. Correct here is in terms of both the database and user semantics. The selection operation is a straightforward process, given the conditions of qualifications for the selection.

The specification to derive the required view structure must be semantically-equivalent and consistent. Used here, semantically-equivalent means the correct matching of user's intended semantics with the definition of the view semantics in terms of database objects. The requirement of equivalence and consistency is necessary for any mapping facility; it is especially critical here, given that a main objective of a view here is to close the semantic distance (the term borrowed from Nijssen 77) between the database and the end user.

Notably, the specification of relationships between database entities does not necessarily form an appropriate access path for the actual retrieval of data values for an occurrence structure. Such access paths will often depend on optimisation strategies, or on the particular form of query posed by the user on a given query.

3.1.6 Summary

The five functionalities discussed above are necessary elements of a mapping facility for the overall transformation of database structures to a query view. The aspect of semantics has been intuitively implied in the descriptions; its abstract notion has made it difficult to confine it solely to a specific area of discussion. The extent of semantic changes and structural differences are especially highlighted in the following section, in the range of data type transformations that is used to create a linear repeating view structure and its associated occurrence structure.

3.2 Ways of transforming : a classification

The transformations that are contained here are from the entities, attributes and relationships of a conceptual schema to data items and view structures in an external schema. Aspects of name and format changes are assumed to be part of the overall mapping.

3.2.1 Data item transformation

This sub-section looks at the various ways data items in the view may be specified with the possible corresponding data value changes. The categorization is split up into two parts : Section 3.2.1.1 examines changes from database attribute types --- these are commonly encountered; Section 3.2.1.2 features derivation from non-attribute source types i.e. from database entity and relationship types --- the significance of such transformations is the adaptation of one form of data description and its role as an attribute or entity or relationship type as defined in the database conceptually, to suit different user's perspective.

3.2.1.1 Conventional

o The simplest class of transformation is a direct mapping from an attribute. The data value remains the same but the name and format may change if required.

o The second class of transformation involves an algorithmic computation using the values of one or more database attributes, to give the transformed value for a single data item. The different types that fall into this category can be grouped according to the kind of algorithm used:

-- An arithmetic expression computes a transformed value from occurrences of related attributes associated to the same or possibly different entity type. Parameters with values supplied by the EUA or even the end user may also form part of the expression.

-- A table lookup is used to convert values of an attribute to a semantically-equivalent value for the corresponding data item. A nice example due to Date 77 is where a database attribute COLOR has values represented as integers. The equivalence of these values can be coded in a table such that 1='RED', 2='BLUE' and so on. Another application is in the conversion of units for numeric values of attributes. For instance, the database attribute SIZE of say, SHIRT, may be represented in 'inches'. These values may be coded by prior evaluation of an arithmetic expression to values in 'centimetres' that are entered into a table.

-- A standard summarizing function produces a single data item from the set of occurring values of a particular database attribute type. Examples of these functions are TOTAL, COUNT, AVERAGE, MAXIMUM and MINIMUM.

o The third class is an aggregate mapping from values of more than one database attribute types. The value of the data item is a combination of the values of semantically associated attribute occurrence values. The relevant occurrences all belong either to one database entity occurrence, or to one such occurrence and its associated occurrences of closely related database entity types, such as the owner of its Codasyl set.

3.2.1.2 Semantically-motivated mapping

The term **semantically-motivated** mapping refers to the means of representing a piece of data as a different semantic category of information from the underlying human perception of the data. Categories of information and semantics in this context, is expressed in terms of attributes, entities and relationships. The capability to project the type of role a piece of data assumes in differing user's perspective, has been referred to as semantic relativism [Brodie 79, Hammer and McLeod 81, Kreps 80]. Significantly, it forms one of the fundamental aspects of semantic data modelling.

In general, classical structured data models such as the relational, network or hierarchical model lack this necessary flexibility of viewing the roles of data differently. This of course is one major criticism underlined strongly by semantic data model advocates. Indeed, we believe that conventional data model-based systems must provide semantic-motivated mappings in order to make up for the inherent limitation in this aspect of capturing real-world semantics. It also follows that in using a semantic data model at the conceptual level, the need to support semantic-motivated mappings, interestingly becomes secondary; the database theoretically holds various semantic forms of data and hence it should be possible to employ a direct mapping instead.

Three common and useful classes of semantic-motivated mappings are identified as follows:

o Mapping of an entity type to a data item value.
  This takes a database entity type name and treats it as a value for a mapped data item in the view. Again returning to the HOSPITAL database example, suppose a general class of view object denoted by the view HOSP-STAFF, is created from the database entities NURSE and DOCTOR. Each occurrence of the view HOSP-STAFF can be a NURSE or a DOCTOR occurrence.
  It may be useful in such a transformation, for the end user to be able to identify whether data about a HOSP-STAFF is indeed that of a NURSE or DOCTOR. For this, a new data item JOB-TITLE can be included in HOSP-STAFF, the value of which is derived from one of the two entity names, DOCTOR or NURSE.

o Mapping of a relationship type to data item value.
  In a similar way, the name of a database relationship (type) can be regarded as a data item value in a view. To illustrate this, let us assume that the database entity DOCTOR attends two types of patients

i.e. OUT-PATIENT and IN-PATIENT. A view about DOCTOR and all the related patients (both OUT and IN-PATIENTS) can be defined as follows:

DOC-PAT (D-name (P-no , P-name)) , where P-no and P-name together form a repeating group. It may be desirable in many circumstances, to preserve the information about the patient's category in the view. This can be achieved by mapping a new data item called CATEGORY associated with each PATIENT, whose value is taken from the relationship name, OUT-PATIENT or IN-PATIENT.

- The third class of semantic-motivated mapping abstracts the description of a database attribute into an explicit new data item type in the view. The part of the database attribute name which describes the characteristics of the attribute becomes the value of the data item type. The mapping is reversible i.e. the occurrence value of a database attribute type becomes part of the name of a data item type i.e. it is used in the semantic naming of a data item.

These types of mappings generally occur alongside the structural transformation of linear structures to a repeating structure in the first case, and that of a repeating structure to a linearised form in the second case. The usefulness of these mappings will be fully illustrated in Section 3.2.2.5.

To sum up this section then, semantic-motivated mappings (in the context of conventional database systems) are generally used in conjunction with an overall transformation of the whole view structure. The motivation is to preserve the original semantic contents that helps to enhance the view's intent, thus making it more easily understood.

3.2.2 Data structure transformation

The above discussion describes transformations of data item and its value. This section now considers the reorganisation of database structures into a single view structure. Two basic forms of changes can be defined: an **intra-structural** change is defined to be the reorganisation of attribute types and any or all of their occurrences within one database entity type structure. It is less explicit in its structural manifestations i.e. no obvious modification to the overall structure, but encompasses a distinct change in the semantics and hence the use of the view.

An **inter-structural** change defines a reorganisation of data at a higher level of framework i.e. among entities in a larger context. The restructuring work of Dale and Dale [76,77] is one general example of inter-structural change.

Based on these two concepts, five classes of structure transformations are identified. The first three signify an inter-structural change, while the latter two describe intra-structural change.

3.2.2.1 Concatenating transform

It is useful, first of all, to understand the context of such transformations. A loose structure is defined in terms of **nodes** and **arcs** which represent database entities and relationships respectively. The relationship between two adjoining entities can be expressed by an immediate arc. This refers to a single linkage between two nodes with no other nodes in between.

An immediate arc can be written as: $A(N_i, N_j)$
The equivalent diagrammatic representation is N_i —— N_j denoting a 1:1 relationship, or $N_i \longrightarrow N_j$ denoting a 1:n relationship. It is further assumed that complex m:n relationships are resolved into two separate 1:n relationships in the database.

The **concatenation** of a set of database entities E_1, E_2 ... E_i denoted by nodes N_1, N_2 ... N_i in a loose structure, is defined to be the directional collapsing of the associated immediate arcs between the specified nodes, such that the full set or a subset of the nodes i.e. the database entities, form a contiguous linear structure. The concatenation must consist of one origin node; this is defined as the centre point from which the collapsing of immediate arcs is propagated. (Origin node here represents the base entity of the view thus transformed).

The direction of the collapse of an immediate arc is represented by imposing a shaded arrow as follows: $N_i \relbar\!\blacktriangleright\!\relbar N_j$. In terms of the concatenated occurrence structure, each occurrence of the node from which the collapse is directed (N_i), may be associated to one or multiple occurrences of the 'destination' node (N_j). The number of occurrences associated depends on the degree of the relationship the arc bears. In the case of a 1:1 relationship, there will always be only one occurrence of the destination node for each occurrence of the 'source' node. With 1:n relationships, two different situations arise and are drawn as follows:

i. $N_i \relbar\!\blacktriangleright\!\!\longrightarrow N_j$

The relationship between N_i and N_j is 1:n. The direction of the collapse, in this case, from N_i to N_j has a one to many context. (This is denoted by the shaded arrow pointing towards the multiple relationship indicated by the single arrow). Collapses of this kind will result in one occurrence of the source node related to multiple occurrences of the destination node.

ii. $N_i \relbar\!\blacktriangleleft\!\!\longrightarrow N_j$

The direction of this collapse this time is from N_j to N_i. The relationship between N_j and N_i is n:1, i.e. each occurrence of N_j is associated to one occurrence of N_i. Exemplary collapse of this generic form, in a many to one context, will result in one occurrence of the source node associated to one occurrence of the destination node.

The direction of the collapse, thus significantly determines the corresponding occurrences for a given concatenated linear structure.

On the whole, concatenating transforms represent the majority classes of complex structural transformations. Three cases of variants are presented here. In the following diagrams, these other notations apply:

o nodes of database entities are written as capital letters e.g. Y. With the concatenated structure,
o brackets around a capital letter denote multiple occurrences of the corresponding entity; if otherwise, a single occurrence is implied.
o in each case, X represents the origin node in the schema diagram, and the equivalent base entity in the view structure diagram.

Case I : a loose hierarchical structure

An example:

$$\begin{tikzcd}
X \arrow[d, two heads] \\
Y \arrow[d, two heads] \\
Z
\end{tikzcd}$$

=> i. X (Y (Z))

ii. X (Z)

Intermediate nodes may be omitted as in (ii). Navathi and Fry 80 in their work, used the term compression to refer to the same class of restructuring.

Case II : a loose network structure

Example i.

$$\begin{tikzcd}
X \arrow[r, two heads] \arrow[dr, two heads] & S \arrow[r, two heads] & Y \arrow[dl, two heads] \\
& Z &
\end{tikzcd}$$

=> X (Y)

This example illustrates the important requirement of identifying the correct view occurrence structure. The concatenation of nodes X, Y may be accomplished by two different sets of arc definitions:

a. A(X,S) or b. A(X,Z)
A(S,Y) A(Z,Y)

Example ii.

$$\begin{tikzcd} X \arrow[dr] & & Y \arrow[dl] \\ & Z \arrow[d] & \\ & S & \end{tikzcd}$$

=> X (Z Y (S))

Example iii.

$$\begin{tikzcd} & X \arrow[dl] \arrow[dr] & \\ Z \arrow[d] & & Y \\ S & & \end{tikzcd}$$

=>

i. X (Z (S)) (Y)

ii. X (Z) (Y)

Example iv a.

$$\begin{tikzcd} & X \arrow[dl] \arrow[dr] & \\ Y \arrow[rr] & & Z \end{tikzcd}$$

=> X (Y) (Z)

Example iv b.

$$\begin{tikzcd} & X \arrow[dl] \arrow[dr, twoheadrightarrow] & \\ Y \arrow[rr, leftrightarrow] & & Z \end{tikzcd} \quad \Rightarrow \quad X\ (\ Z\ (\ Y\ )\ )$$

Examples (ii) through (iv) show the wide range of possibilities.

3.2.2.2 Merging transform

The merging of two or more database entities belonging to the same set of classification, is defined to be the creation of a single new object that represents the common class. The domain of data occurrences of a merged data object is drawn from the selected set of data occurrences of the source entity types. The qualification of source data occurrences is optional; it is achieved by matching attribute value in an occurrence, with the specified qualifying conditions.

A merging transform is one way of conceptualising new semantics. The earlier example of the generalisation of HOSP-STAFF is one practical illustration of a merged view structure.

The figure below shows the mapping of source occurrences to the merged object occurrences.

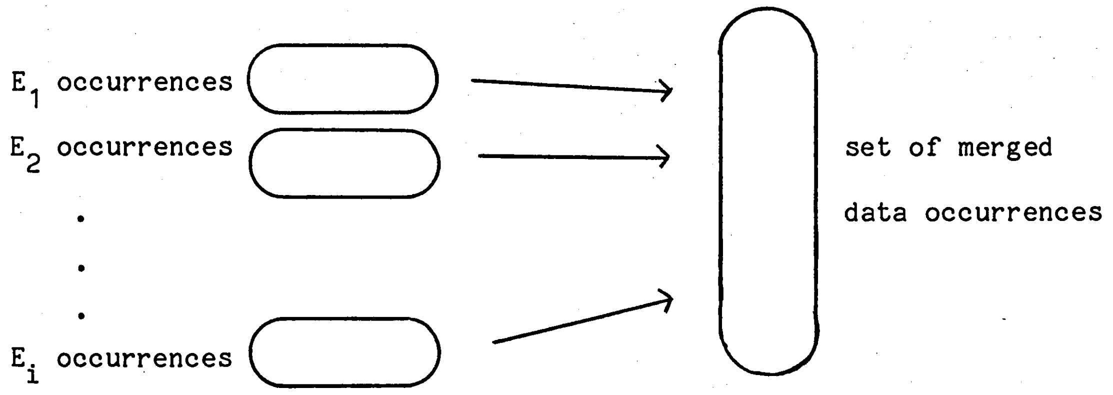


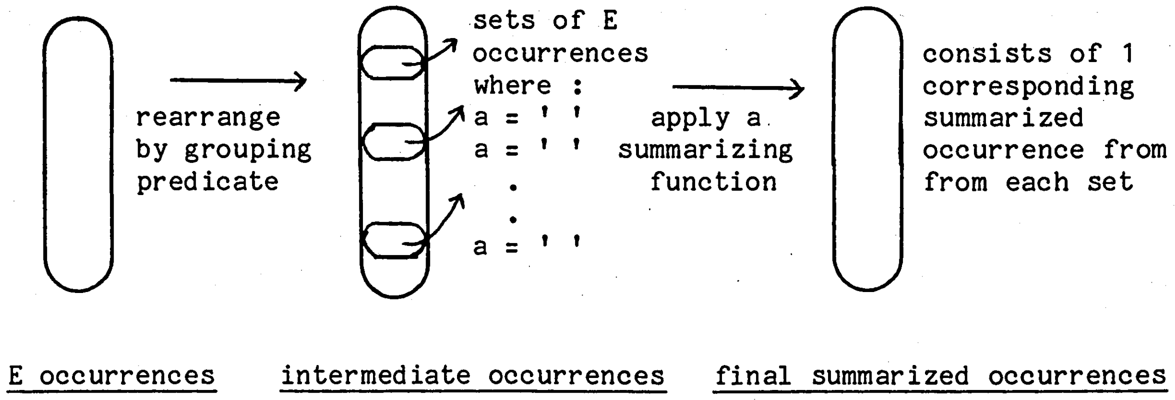
fig. 3.1 Merging Transform at the occurrence level

3.2.2.3 Subsetting transform

A subsetting transform defined on a database entity type results in the formation of a new semantic object with data occurrences which satisfy the qualifying conditions. The requirement of data occurrence selection is the basis of this type of transform. A thorough treatment of the semantics, and an example have already been given in Section 3.1.4.

3.2.2.4 Summarizing transform

A summarizing transform must be specified with a named 'grouping attribute'. This must be a valid attribute of the entity, on which the transform is applied. When defined on an entity, it reorganises the data occurrences into sets of occurrences based on a common value of the grouping attribute. In order to produce the summarized content, a summarizing function e.g. AVERAGE, is used to aggregate the required data values in each set. The semantics of the transformed object is chiefly characterised by the grouping attribute. The number of summarized data occurrences is equal to the number of unique values of the grouping attribute in the source occurrences. The effects are illustrated as follows:
A database entity with its attributes is written as E ($a_1, a_2 \ldots a_i$)
where a_1 is the grouping attribute.



rearrange by grouping predicate
sets of E occurrences where :
a = ' '
a = ' '
a = ' '
apply a summarizing function
consists of 1 corresponding summarized occurrence from from each set
E occurrences
intermediate occurrences
final summarized occurrences

fig.3.2 Summarizing Transform at occurrence level

3.2.2.5 Transposing transform

The transposition of a database entity reorders occurrences belonging to the same type or having a common property, in a non-repeating linear flat structure to a repeating data structure of occurrences, or vice versa. In general, the transform is defined in conjunction with a concatenation of related entity types in a loose structure framework.

It must be emphasised that in the following cases, the chief interest is in the kinds of semantic derivations and differences in structures that arise from the transformation. Details of the exact means of specifying such a transform are not a concern here. The notations used in Section 3.2.2.1 apply here as well.

Case I : Simple Linear to Repeating Transform

$$\boxed{X \longrightarrow} \; Y\,(\,A'_1 \; A'_2 \ldots A'_3) \quad \Rightarrow \quad \boxed{X \text{---}} \; Y\,(\,A'\,)$$

A' named within brackets represents attribute type.
A dotted square box indicates the presence of related entity types.
The corresponding occurrence structure is as follows:

$$\boxed{x \longrightarrow} \; Y\,(\,a'_1 \; a'_2 \ldots a'_i\,) \quad \Rightarrow \quad \boxed{x \text{---}} \; Y \begin{pmatrix} a_1' \\ a_2' \\ \cdot \\ \cdot \\ a_i' \end{pmatrix}$$

flat linear structure — repeating group

Lower case letters a' represent data values that do not necessarily differ, of the corresponding attribute type A'.

Several things are to be noted with this case:

1) The transposition involves significantly a merging of attribute types of a similar class to a generalised attribute type i.e. A_1' A_2'... A_i' to A' in the view.

2) As with merging transforms in general, some original information may be lost in the process.
The semantics of the set of source attribute types A' can be broadly categorised as:-
   - o Consisting of explicit descriptive characteristic or role attached to each element of the set e.g. <u>Eye</u>-test-result, <u>Skin</u>-test-result and <u>Blood</u>-test-result => (General) Test-result.

   - o Pertaining to an implicit ordering sequence e.g. Test<u>1</u>, Test<u>2</u> and Test<u>3</u> => Test.

The generic figure above illustrates the latter kind. In this case, the end user may intuitively perceive from the repeating structure of occurrences in the view, the possible ordering sequence in which the data values a' appear, which can be reflective of the source semantics. This is one demonstration of how a data structure in fact captures certain semantics. (See subsequent example).

With the first category, information about the individual roles of the source attributes will be submerged in the transformation unless these characteristics are abstracted out explicitly by a semantic-motivated mapping as described earlier. This type of semantic-motivated mapping is less clear-cut than the previous ones. It involves the partial extraction of the source attribute type name to become the data value of the newly-created attribute type in the view. To illustrate the point, the first example is expanded:

A semantic-motivated mapping is defined such that a new attribute type called 'Type' (of test) is created to describe the corresponding attribute type 'Result'. The domain of values of 'Type' is then drawn from the characteristic description of source attribute type names, namely, 'Eye, Skin and Blood'. Hence, instead of just seeing a broad heading of (general) 'Test', each is now broken down into specific ones expressed by two attribute types 'Type' and 'Result'. The graphic representation is as follows : X in the generic figure is now replaced by Patient with attributes P-no, P-name; R abbreviates Result. For simplicity, only the occurrence structure is shown.

```
P-no  P-name  Labtest(EyeR  SkinR BloodR)    P-no  P-name  Labtest( R )
123    Jones           +ve    -ve   +ve    => 123   Jones            +ve
                                                                     -ve
                                                                     -ve
```

Applying a semantic-motivated mapping on the transposed view, it becomes:

```
                                  => P-no  P-name  Labtest(Type, R)
                                     123    Jones        Eye    +ve
                                                         Skin   -ve
                                                         Blood  +ve
```

The example brings out a further point. Such a transposition is useful for the purpose of suppressing particular information; this may be one way of restricting complete access to certain types of information. On the other hand, full preservation of semantics can be achieved by means of semantic-motivated mappings.

Similarly, a graphical illustration of the example in the second category is now given. The implicit ordering is explicitly defined using a data item Seq-no (Sequence-number) in the view.

```
P-no  P-name  Labtest(R1  R2  R3)    P-no  P-name Labtest(Seq-no  R )
123    Jones          +ve -ve +ve => 123    Jones             1   +ve
                                                              2   -ve
                                                              3   +ve
```

<u>Case II : Repeating to Linear Transposition</u>

This is just an inverse of Case I. The general representation is:

$$\boxed{X{-}{-}{-}>}\ Y\ (A') \qquad \Rightarrow \qquad \boxed{X{-}{-}{-}}\ Y\ (\ A_1'\ \ A_2'\ \ldots\ A_I'\ )$$

The occurrence structure is:

$$\boxed{x{-}{-}{-}>}\ Y\begin{pmatrix} a_1' \\ a_2' \\ \cdot \\ a_i' \end{pmatrix} \qquad \Rightarrow \qquad \boxed{x{-}{-}{-}}\ Y\ (\ a_1'\ \ a_2'\ \ldots\ a_1'\ )$$

<u>repeating group</u> <u>flat linear structure</u>

The linearization is obtained by generating a new attribute type that corresponds to one occurrence of the repeating structure, for every occurrence that exists. In this process, the source semantics modelled implicitly or explicitly, must be captured in the naming of data types in the view in order that the view be meaningful. Consider the following example which is a complete opposite of the one above.

P-no	P-name	Labtest(R)		P-no	P-name	Labtest(R1	R2	R3)
123	Jones	+ve	=>	123	Jones	+ve	-ve	+ve
		-ve						
		+ve						

Here, the repeating occurrences of Labtest are transposed to a linear form. The inherent structural information of the source occurrences is expressed by the names of R1, R2 and R3. Note that there is no reason why these names should not simply remain as R, except that such a view will not be very meaningful to be of good use.

Depending on how the source structure is modelled, a semantic-motivated mapping may be required, as in the following sequel:

P-no	P-name	Labtest(Type,	R)		P-no	P-name	Labtest(EyeR	SkinR	BloodR)
123	Jones	Eye	+ve	=>	123	Jones	+ve	-ve	+ve
		Skin	-ve						
		Blood	+ve						

In this instance, R is characterised by another attribute type, 'Type'. It is essential that this information should be preserved in the view for a clear understanding of the semantics. The semantic transformation that accompanies this class of transposition is from a data value to a conjunctive attribute type name i.e. the data value only forms the descriptive part of the name, which has to be attached to a main attribute (in this case, R).

TABLE I Classification of Data Transformations

TRANSFORMATION	CLASSES		
Data Structure	Concatenating transform		
	Merging transform		
	Subsetting transform		
	Summarizing transform		
	Transposing transform		
Data Item	Direct value mapping		
	Conventional	Algorithmic	Arith. Expr.
			Table Lookup
			Summ. Function
		Aggregate value mapping	
	Semantic-motivated Mapping	Entity-type to data item value	
		Relationship type to data item value	
		Attribute characteristics to data item type	

3.3 Form of mapping specification

We consider two important aspects of a mapping specification:

- o the provision of a language that supports the different classes of transformations. This language will be referred to as the mapping definition language, and
- o the mapping specification process.

3.3.1 The mapping definition language

A mapping definition language desirably, must be simple in order to be employed easily, but yet be sufficiently powerful to encode the required complex transformations as well.

Two interrelated issues are central to the properties of a language: its syntactic structure and the degree of procedurality. The syntactic structure refers to how the syntax of the language is built. Codasyl-based languages such as the Codasyl DDL, DML are distinctly characterised by a separation of different functionalities into individual sections. Relational mapping languages such as SQL support self-contained expressions within a single statement. The syntactic structuring of languages imply:

1) a set of logical operations that may not necessarily be in the desired semantic order. For instance, in the Codasyl EUF DDL syntax, the item conversion specification is such that it appears before the specification stating how these items are extracted from the source types. The correct semantics requires that the extraction be carried out prior to any transformation. The recognition of the right order of logical operations is however, usually performed by the compiler [Leavenworth and Sammet 74].

2) a sequencing of how requirements must be specified in order to achieve the desired results. This is a question of procedurality (or non-procedurality). Obviously, as a measure of independence from the system's implementational and structural constraints, the language should allow one to state what is wanted rather than how to go about doing it. In other words, the language should be as declarative as

possible. There is of course the danger of not knowing if a set of declarative codes will produce the correct required results. However the problem is equally true of procedural languages as well, where syntactically-faultless specifications may also lead to incorrect retrievals of data. This is more of a semantic issue and not a syntactic one. One way to alleviate this is to monitor the interpretation of user's intended semantics with what is actually held in the database. This issue of semantic equivalence and correctness is the prime concern of the next section.

3.3.2 The mapping specification process

There are three general approaches towards definition of views in an external schema:

- o By embedding mapping codes in application programs -- this is the case with Codasyl subschemas.
- o Direct specification -- this is true of definitions by means of query modifications as with the relational systems of System R and Ingres.
- o Automatic generation of subschemas by a program as in Goldman [79].

The last case relies totally on the system to produce consistent derivations. In the other two cases, the user is informed of any inconsistencies that may arise. Validations, however, are mainly for syntactic correctness and include only basic checks of semantic integrity e.g. the format of say, AVERAGE-WEIGHT cannot be a character string. In general, there is no proper means of finding out if the derived data values are the expected ones; erroneous information may be used without the user even being aware of it. As such, this question poses a serious threat to the use of a database and its data integrity.

The approach hereby proposed is bent on the motto of ' Prevention is better than cure '. The issue of semantics is highly intuitive. It is imperative that the process of specification is an interactive one during which the user is:

-- guided to formulate data requirements within the allowed database constraints,

-- made aware of any semantic ambiguity that may lead to wrong derivations and hence knows properly what the ultimate derived values stand for.

In the review of universal relation interfaces in Chapter 2, it has already been pointed out that the heavy dependence on the system to interpret what the user wants by rules stored about the data dependencies, is just as error-prone.

Until a formal and reliable way of validating the semantic consistency of a view with the corresponding human interpretation can be found, it is sensible to minimise the possibilities of such inconsistencies happening.

Chapter 4 RECURSIVE STRUCTURE TRANSFORMATION

There are numerous cases of recursion that crop up in the modelling of real-world applications. Perhaps, by far the most common is the Bill-Of-Materials (BOM) in the manufacturing industry. Others include the 'family-tree' where a person has children. Hierarchies of departments with sub-departments, employers with employees are typical ones found in a large organisation.

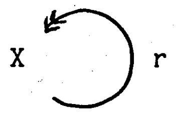
The abstraction into amenable view structures containing the required data from logically recursive constructs in the database represents an important aspect of view transformations. The associated complications have not yet been extensively researched. This chapter attempts to apply the concepts of transformation described in the previous chapter to treat the problem of mapping recursive structures. The approach departs from all other more traditional lines of specifying special retrieval functions. It instead draws on the representation of a view and its semantics so that end users may use a set of simple direct retrieval commands consistent with non-recursive applications, without having to resort to special operators or procedures.

A formal notation to represent the recursive nature of data structures in order to enable consistency of presentation throughout the chapter will first be described.

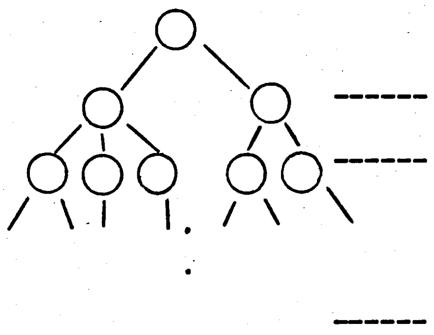
4.1 An introduction to the problem

A recursive data structure involves only one entity type whose occurrences inter-relate in 'loops', the number of recursive levels of which cannot be fixed a priori. A recursive structure therefore can be more easily appreciated by studying its occurrence structure. This can be best represented in terms of a recursive hierarchical tree (c.f. Clemons 81) whereby the nodes of the tree denote data occurrences of the same entity type.

First, consider the basic form of a single recursive loop structure. This has unique data occurrences linked by one kind of relationship. In general, such a structure will be represented as:



where r is the recursive relationship defined on the entity type X. Due to the arbitrary degree of recursion that can exist with such structures, it is difficult to present a generalised hierarchical occurrence structure accurately. A typical one will probably be:



N_1 is the set of nodes at level 1

N_2 is the set of nodes at level 2

.

.

N_i is the set of nodes at level i

fig. 4.1 A Recursive Occurrence Structure

In order to distinguish clearly the use of the term node in a loose structure (See Section 3.2.2.1) from this present context, a node of entity occurrence in the tree structure will instead be referred to as <u>occurrence-node</u> (o-node) hereafter.

An o-node is drawn with a symbol ○ . The collection of o-nodes at a particular level i are denoted by N_i . Using the normal conventions for describing a tree hierarchy, each o-node rests on a particular level which is its depth in the hierarchy from the topmost root. O-nodes lying on the same level are said to belong to the same level set (of occurrences). Those which have no further sub o-nodes hanging from them are known as leaf o-nodes.

Based on the above descriptions, three problems encountered in the mapping of recursive structures are identified as follows:

1) The depth from the root o-node to each of the leaf o-nodes in any given occurrence structure varies and cannot be predetermined. Currently, most existing query languages do not provide adequately-general specification facilities for end users to retrieve information from recursive structures, without having to first define a specific depth of nestings of retrievals. All relationally-complete languages, except QBE [Zloof 75, 76, 77] fail even to resolve this inherent recursive problem.

2) The position of an o-node as denoted by its level (number) embeds essential information about the real-world semantics. For instance in the case of modelling generations of 'ancestors-descendents', the level of a given o-node from the root of the hierarchy expresses implicitly the generation (number) of the corresponding descendent-person to the ancestor-person represented by the root. In general, this kind of information will be lost in a retrieval process, unless it can be explicitly abstracted. In applications where such information is crucial to the processing needs, for example the different 'levels' of components used in the manufacture of a part, it is common to assign an explicit attribute to the component entity which describes the level-number of the component. This simplifies the data processing involved and ensures that the essential information will be available.

3) Whilst the above two problems concern the extraction of data, a third problem relates to the representation of retrieved data in unit view structures for end users.

A recursive structure may be regarded as having a 'breadth' and a 'depth' [Bobrow 71]. (Bobrow used the term width instead of breadth). The breadth represents all o-nodes at an immediate level belonging to a higher o-node. The depth represents all o-nodes which relate to a given o-node, at each of the various levels. Repeating groups can be used to represent in a view, syntactically the arbitrary breadth or depth of a recursive structure. The complete semantic content though, cannot be captured.

Each data occurrence associated to an immediate parent occurrence, often always relates to other data occurrences at the immediate lower level.

Presently, there is no general mechanism for structuring all recursive levels of information into one view structure such that these inter-relationships between the data occurrences are fully preserved. To illustrate this, consider the recursive structure drawn in fig. 4.1a below. The above statement means that it will not be possible to have an equivalent view structure such that the end user can perceive directly and extract the following information from the view -

i) all occurrences related to a -- b, c, d, e, f, g, h, i, j, k, l and m, belong to three different levels.

ii) and that b itself relates to, i.e. has 'children' e, f
c itself relates to, i.e. has 'children' g
d itself relates to, i.e. has 'children' h, i and j

iii) and that f in turn relates to k,
h in turn relates to l and m.

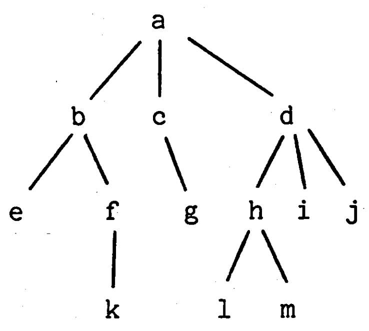


fig. 4.1a Example of a Recursive Structure

Very little work has been devoted to investigating the above three described areas. The two significant pieces of contribution so far, by Zloof and Clemons [76b, 81] have concentrated mainly on obtaining the required data values without paying sufficient attention at all to the overall semantics of the information that can be derived. Although Clemons advocated for user-oriented data structures in application interfaces to treat recursive examples, he looked mainly at two issues:

o the extraction of processed data from recursive database relations, and other related data held in different relations.
o the extraction of processed data from the recursive structures.

The latter facility is supported by external functions defined to retrieve certain portions of the recursive structure with the main aim of computing derived values for very specific application purposes. The functions, as such, are not quite general enough. There is also very little effort paid to the abstraction of meaningful information about the basic relationships between sets of occurrences embedded in the self-nested structure, into semantic user views. Nevertheless, the work represents a very useful achievement towards the direct retrieval requirement of a view identified in this research here.

The remaining presentations here will specifically concern the structuring of user views from recursive structures, in ways that can be general-purpose. It will be shown step by step how concepts of transformation, namely concatenating, transposing and semantic-motivated can be used in combination to produce meaningful view structures in an attempt to eliminate the problems mentioned above. Any further processing of data values can then be defined on top of these view structures by means of an item transformation as listed in Section 3.2.1.

4.2 Linear view structuring

We first consider the transformation of the data occurrences in the recursive hierarchy in fig. 4.1 into data occurrences held in a unit linear view structure. An appropriate transform for the purpose will be a concatenating transform. However, in order to see how this may be applied, the general context of concatenation in a loose structure of database entities and relationships must be broadened to treat a framework of entity occurrences and relationships of one type.

A concatenating transform will now be defined on o-nodes instead of nodes. An immediate arc between two o-nodes represents the selfrecursive relationship, and is depicted as: $$\begin{tikzcd} \underset{n_i}{\bigcirc} \arrow[r, "a"'] & \underset{n_j}{\bigcirc} \end{tikzcd}$$,

where the shaded arrow again represents the direction of collapse. For the time being, only single recursive loops with a 1:n relationship, is considered. Notice here that the effect of the collapse-direction in the sense described in Section 3.2.2.1 on the occurrence structure formed, is no longer relevant because the collapse now operates explicitly on the data occurrences themselves. It is also important to distinguish a root o-node and an origin o-node. A root o-node is always the top-most single o-node in the hierarchy and is not necessarily an origin o-node from which the collapse propagates.

The way that the real-world semantics of recursive applications are captured in database structures requires different ways of collapsing in order to extract specific types of query information. In general, there are two ways in which the immediate arcs may collapse.

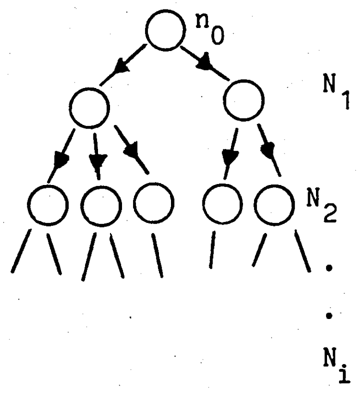
Definitions

A breadth-wise collapse is defined to encapsulate the positional semantics that is formalised by the term level, embedded in the tree occurrence structure. The collapse operates across a set of o-nodes belonging to the same level with respect to an origin node. It only extracts information pertaining to o-nodes directly related to one another in a given level of the tree hierarchy.

A depth-wise collapse is defined to abstract the relationship of o-nodes lying along a non-branching, hierarchical path in the tree occurrence structure. The collapse only extracts information pertaining to o-nodes directly related in only such a hierarchical path of the tree hierarchy.

The graphic representations of these two collapses are shown in figs. 4.2 and 4.4 respectively. Accordingly, the transformed figure drawn on all the following diagrams depict occurrences and not types, i.e. it shows the occurrence structure of the view.

4.2.1 Breadth-wise extraction of semantics

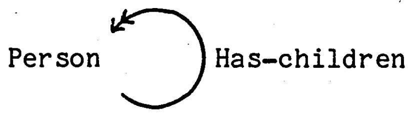


N_1 is collapsed to become $n_0 \{N_1, N_2, \ldots N_{i1}\}$

fig. 4.2 Breadth-wise collapse

In the diagram, n_0 is the origin node. { } are used to indicate the association of data occurrences to the origin o-node occurrence.
If we regard the set of o-nodes, N_1 to N_i as generically the 'members' of 'owner' origin node, n_0 , the semantics of the collapsed occurrence structure can be interpreted as all the members belonging to the specified owner. The following gives a practical example to illustrate the usefulness of a breadth-wise collapse. In such cases, qualifying conditions are always used to select that subset of an occurrence structure identified always by a specified root o-node, on which the transform is applied.

In the first instance, a single recursive loop is used. For the purpose, the case of family-tree is assumed to be a single recursive loop. The recursive data structure is represented as :



where Person is the database entity and Has-children is the self-nested relationship. The Person entity will have attributes Person-name, Age, Sex, etc. ; for simplicity, occurrences of Person will only be identified by the value of the attribute Person-name, e.g. John is an occurrence of Person entity.

Let us now consider the breadth-wise collapse of a subsetted occurrence structure of the family tree, corresponding to the generic fig. of 4.2. This is drawn as follows, with Joe as the origin node. Note that the generic symbol of an o-node is now replaced by the identifying name of the entity occurrence in question.

$$\begin{tikzcd}
 & & \text{Joe} \arrow[dl] \arrow[drr] & & & \\
 & \text{Mary} \arrow[dl] \arrow[dr] & & & \text{John} \arrow[dl] \arrow[d] \arrow[dr] & \\
\text{Jim} & & \text{Pat} & \text{Ben} & \text{Jill} & \text{Bill}
\end{tikzcd}$$

=> Joe {Mary, John , Jim, Pat, Ben, Jill, Bill}

fig. 4.3 Family-tree example

In order to represent the above collapsed occurrences in a unit view structure, one must be able to refer to the sets of occurrence values of Person-name generically as typed data items in the view, that convey the view semantics. Identifying the roles of the occurrences in this example, the members are the descendents (Mary, John, Jim, Pat, Ben, Jill and Bill), whilst the owner is the ancestor (Joe). Obviously, descendents repeat within ancestor. It is therefore appropriate to transpose the flat representation into the following view structure :

Ancestor	(Descendent)
Joe	Mary
	John
	Jim
	Pat
	Ben
	Jill
	Bill

The database entity type of Person has now been abstracted into two distinct roles of Person, represented by data items Ancestor and Descendent, that inform the end user the semantics of the application.

There are two problems of semantic preservation though. First of all, the fundamental piece of information that values (of Descendent) belong to different levels in the original hierarchy is lost. All the descendents here are simply lumped together as one whole 'mass'. The fact that Mary and John are related as immediate descendents and that the rest belong to the second generation has vanished.

The second point is that the inter-relationships between the descendents themselves are also hidden away. For example, Jim and Pat are further associated to Mary as her children. (Recall the third limitation identified on page 50 of not being able to capture the full intent of the data in a unit view).

In order to resolve the first problem, a semantic-motivated mapping is applied. Just as in Section 3.2.2.5 where a semantic-motivated mapping is used to project out an explicit type characteristic or a piece of 'ordering sequence' information in the database structure, a semantic-motivated mapping may be applied here to project the original semantics. The position of the level of an occurrence in the hierarchy may be extracted out by means of an externally-defined procedure. Such a procedure is required because unlike earlier examples of semantic-motivated mapping, there is no explicitly stored assignment of a value for level number as such (unless as mentioned before, the information is modelled explicitly as an attribute). Thus, in this family-tree example, a new data item called Generation, can be derived with values indicating the number of the generation, as shown below.

	Ancestor	(Generation	(Descendent))
=>	Joe	1	Mary
			John
		2	Jim
			Pat
			Ben
			Jill
			Bill

Notice that this does not alleviate the second problem i.e. the occurrence structure is only able to convey the information that Jim, Pat, Ben, Bill and Jill are grand-children of Joe, but does not inform any further that Jim and Pat are direct descendents of Mary whilst the other three belong directly to John. Recall however, that the semantics of a view (in the way that we have defined in Chapter 2, is with respect to the base entity of the view. In recursive examples, it is with respect to the particular data occurrence representing the base in the view; in this case, it is in the form of the data item Ancestor. All other data items in the view, namely, Generation and Descendent therefore characterise the data item Ancestor. Hence in the view occurrence where Joe is the Ancestor, the information that Jim and Pat are descendents of Mary does not directly describe Joe; it relates specifically to Mary and consequently can be obtained by looking at the view occurrence where Mary is the ancestor. No essential information about Joe is strictly lost. For this reason, it is argued that this latter case does not constitute a loss in information in such a context. Rather, it is regarded as a compression of information that cannot be properly represented by the semantics of a view defined by the base entity concept and the subsequent notations adopted for this presentation.

One last interesting observation, though perhaps digressing from the main point slightly, can be made with this example. This regards the useful application of different forms of semantic-motivated mapping and naming to capture specific semantics of an application. Two separate views containing distinct meaning can be derived from the view above. These are as follows :

i) Children (Parent-name , Children-name)

Parent-name	Children-name
Joe	Mary
	John

ii) Grandchildren (Grandpar-name , Grandchild-name)

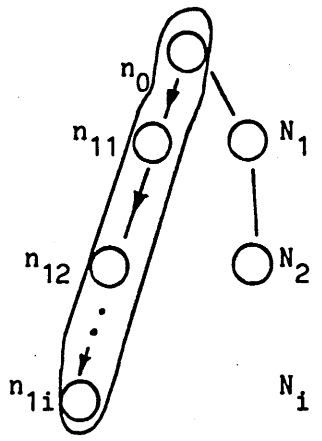
Grandpar-name	Grandchild-name
Joe	Jim
	Pat
	Ben
	Jill
	Bill

The semantics of the data item Generation (i.e. level in the hierarchy) is embodied by the specific name of Children and Grandchildren, i.e. of data objects possessing concepts in the real-world context. The compression of information now disappears.

4.2.2 Depth-wise extraction of semantics

A depth-wise collapse can be further divided into two categories. In one category, the collapse descends downwards with respect to the root o-node of the occurrence structure, from a specified origin o-node in a given path to the leaf o-node. In the second category, the collapse ascends upwards from a specified origin o-node to the root o-node in a given occurrence structure. Both leaf o-node and root o-node are named here for the sake of generality; they can be replaced by an appropriately specified o-node that may be qualified.

In the following diagrams, the data occurrences collapsed are those in the encircled hierarchical path. Suffixes are introduced to o-node in the form of n_{pq} where q is the level number at which the node is located, and p indicates the pth element in a set of nodes at a specified level q.

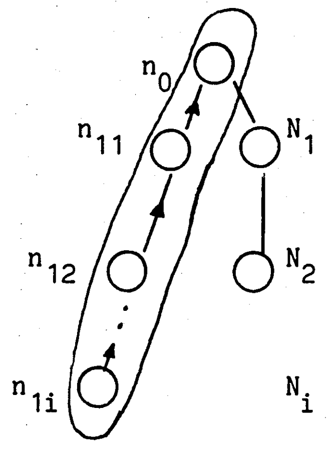


is collapsed to become $n_0 \{ n_{11}, n_{12}, \ldots n_{1i} \}$

n_0 is the origin o-node

n_{1i} is the leaf o-node

fig. 4.4a Descending :



is collapsed to become $n_{1i}\{ n_{1i-1} \ldots n_{11}, n_0 \}$

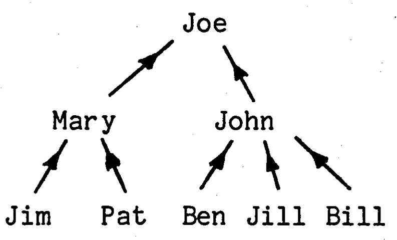
n_{1i} is the origin o-node

n_0 is the leaf o-node

fig. 4.4b Ascending :

fig. 4.4 Depth-wise Collapse

The two forms of depth-wise collapse can be usefully applied in practical cases to extract real-world semantics to give information of a different kind to that in a breadth-wise collapse. Consider again the occurrence structure of the family-tree example in fig. 4.3. In order to produce a view of 'descendent and all his/her ancestors', an ascending depth-wise collapse, is needed this time. The diagram below shows the final transposed view and several of its resultant occurrences.



=>

Descendent	(Ancestor)
Jim	Mary
	Joe
Pat	Mary
	Joe
Ben	John
	Joe

Each one is one individual view occurrence

.
.

With the BOM parts-explosion example, a descending depth-wise collapse may be usefully applied to obtain the basic components (represented by leaf o-nodes) for a given part (a specified root o-node which is also the origin o-node in this case). An asending depth-wise collapse will be useful to find out the line of 'ancestor-parts' that use a given component.

We shall use this parts-explosion example to consider complex recursive loop structures.

4.2.2.1 Complex recursive loops

A complex recursive loop has a m:n relationship between its occurrences. Often, this is resolved by explicitly modelling the roles each occurrence takes, into separate relationship types and introducing an additional 'intersection' or 'link' entity type, as depicted below.

$$\begin{tikzcd} X \arrow[loop right, "r"] & \Rightarrow & X \arrow[d, shift right, "a"'] \arrow[d, shift left, "b"] \\ & & \text{Link} \end{tikzcd}$$

where a, b are the resolved relationships of r

In terms of the recursive hierarchical representation adopted at the beginning of the chapter, the resultant hierarchy now becomes a network. We propose here to represent such cases by allowing the o-node in an occurrence structure to be replicated. The main purpose of this section is to examine the effects of such replication, if any, on the view representation and the required transformation.

With the parts-explosion example, the Link entity thus introduced usually has a specific role and contains data items that describe so. This will probably be of the form :

$$\begin{tikzcd} \text{Part} \arrow[d, shift right, "\text{is-made-of}"'] \arrow[d, shift left, "\text{is-used-in}"] \\ \text{Component} \end{tikzcd}$$

Part and Component belong to the same type of entity. Part may have attributes Part-no, Part-name while Component has attributes Component-no and Qty-used. Again for simplicity, occurrences of Part and Component will only be identified by their domain value of Part-name and Component-name in the following diagrams, e.g. x, p, d, etc. It is further assumed that a concatenating transform with a breadth-wise collapse has already been applied.

The diagram below shows the final view occurrence structure after applying both a transposing transform and semantic-motivated mappings. x is the origin o-node.

$$\begin{tikzcd}
 & & & x \arrow[dll] \arrow[drr] & & & \\
 & c \arrow[d] \arrow[dr] & & & & d \arrow[d] & \\
 & p \arrow[dl] \arrow[d] \arrow[dr] & q & & & p \arrow[dl] \arrow[d] \arrow[dr] & \\
r & s & t & & r & s & t
\end{tikzcd}$$

=>

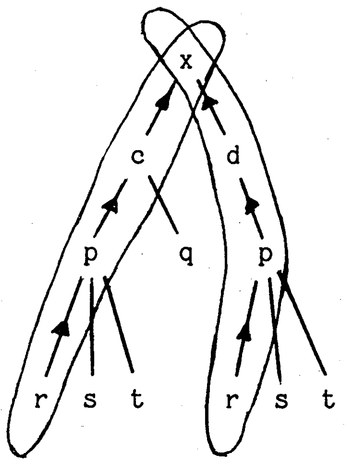
Part	(Level	(Component))
x	1	c
		d
	2	p
		q
		p
	3	r
		s
		t
		r
		s
		t

The meaning of the view here is to give all component types, used in the manufacture of a part, in this case of x. The duplication of Component occurrences in the view occurrence structure -- p at Level 2, r, s and t at Level 3 -- is thus inessential i.e. they do not bear useful information. Furthermore, the presence of such duplication may lead to confused perception of the view by the end user. Accordingly then, these duplicates may be removed. It is interesting though, to note that the duplication serves to highlight the factor of compression of information described in the family-tree example. The first occurrence of p in the view occurrence in fact belongs directly to c, whilst the other p occurrence belongs to d. The same thing happens with the rest of the occurrences. It also demonstrates our earlier assertion that the consequence does not constitute a loss of information, by virtue of the fact that such duplicates may be removed without violating the consistency of the view semantics.

The view occurrence after removing duplicates becomes :

Part	(Level	(Component))
x	1	c
		d
	2	p
		q
	3	r
		s
		t

In a similar way, it can be shown that transformations based on an ascending collapse of concatenating transform may be treated thus. The following diagram gives an illustration. Given the detailed descriptions of previous examples, this one should be fairly self-explanatory. The view occurrence shown is that of component r being the origin o-node.



	Component(Used-in-Part)			Component(Used-in-Part)	
	r	p		r	p
		c			c
=>		x	=>		d
	r	p			x
		d			
		x			

Finally, we caution that the duplication of such occurrences may become essential if further processing of data values is to be defined on top of the view. Consider the derivation of a view with data items : Part(basic-comp , total-qty-used), based on the above existing view. The meaning of the new view is that a given part uses a number of units of basic components and has more than one basic component. Illustrating with just component r in this case, the sub-total of r used along the ascending hierarchical path r, p, c, x must be added to the sub-total of r obtained along the other path r, p, d and x to give the correct final total. In such cases, it is important to preserve all the original number of occurrences in the source view.

4.3 Summary

The basic framework of Chapter 2 has been extended to examine recursive data structures and how information from these structures can be extracted into unit view structures. To this end, two main classes of collapse for a concatenating transform -- breadth-wise and depth-wise, are identified. It is shown how transposition and semantic-motivated mappings can be applied in conjunction, to produce user-oriented view structures.

The transformation semantics and requirements formulated here can form the basis for specifying appropriate operators or functions in the mapping definition language. The derived view structures facilitate simple direct retrieval specifications. It is also possible that the various operators defined with semantics as described in this chapter, be used together in a single view definition to produce more complex views such as the following :

Anc-Desc (Person (Ancestors), (Descendents))

where the Ancestors information is obtained by means of an ascending depth-wise operation and the Descendents information by a descending depth-wise operation.

Chapter 5 AN EVALUATION OF MAPPINGS IN EXISTING SYSTEMS

This chapter studies two contrasting view mechanisms of a. the relational system, System R and b. that proposed by the Codasyl EUFC. These two choices were decided on the grounds that :

o both support the concept of a user view which may be different from the underlying logical schema --
Although the Codasyl EUFC proposals have not yet been implemented, they are nevertheless based on the well-established Codasyl DBMS architecture (which despite not strictly an ANSI/SPARC candidate, has reasonable separation of a user logical schema and database logical schema).

o both (arguably) are directed towards non-expert use, and hence provides a consistent frame for comparison.

o they present two interesting sets of variations in the approach -- System R employs a direct mapping mechanism whilst the second proposal adopts an intermediate path in the form of data items.

The view facilities are evaluated along the mapping framework developed in the previous chapter.

5.1 System R

System R is a relational database prototype system developed by IBM. It is extensively researched into. A commercial product based on this prototype, called SQL/DOS is available on the market. The primary objective of this section is to examine its support for user views. The interested reader is referred to other literature for detailed description of the whole system [Astrahan 75, Date 81].

The fundamental data structure in System R is the base table i.e. a flat structure. A view in System R specifically means a derived table. In the rest of this section, unless indicated by the context of the discussion, the term view will otherwise be a System R view and not that defined in Chapter 1. View definition is by means of direct query modification i.e. in the form of query procedures posed by the users. Constraints though, are exercised if a view is to allow update operations; at this point, only retrieval operations are relevant.

The fact that a view is a tabular structure justifies that some of the criticisms, especially that of Section 4.1.1 are true of the relational model in general. The next section then assesses the impact of a flat table as a view structure. The query language of System R, SQL (Structured Query Language) will be the chief focus of Section 4.1.2; in particular, its mapping power and limitations will be investigated. The user is allowed to define a new view on top of existing views; this has an implication as will be seen in Section 4.1.3. Section 4.1.4 sums up the assessment by answering two questions : a. how far does the System R view match the desired view characteristics set in Chapter 1, and b. what can one then conclude about the view mechanism of System R.

5.1.1 A flat table as a user view

The main point of contention of a flat data representation is its deficiency in capturing and reflecting real-world semantics. The chief and well-publicised drawbacks (see Schmidtand Swenson 75, McLeod 78, Clemons 76b, Codd 79) are :
o no hierarchic structural support for repeating data values
o no semantic data construct for aggregating atomic data values.

These points will be expanded on. It will also be argued how these snags are compounded by the way constraints are dealt with in System R.

A view table in System R is a relation and attributes are called fields. A tabular representation which does not allow data values to repeat, is not at best congenial to end-users' perception, where in many cases, the information content assumes a natural hierarchic structure. In such circumstances, the important concept of a user-perceived real-world object remains hidden. It is more obscured by the fact that a table can contain duplicated rows unlike a strict relation. This is a consequence of the optional constraint on the field types of a view being unique. (System R does not support the concept of keys at all although it provides a unique index system).

Associations between fields cannot be abstracted into a higher level object. There is a complete lack of structured concepts to collect groups of commonly related fields into a semantic data type. As such, the interpretation of a view's semantics relies largely on the intuitive naming of fields. A second source of semantic expression is derived from the (linear) sequencing of field names in a view relation, as displayed to the end user. It is natural that fields which bear a strong association to each other are placed next to each other. Furthermore, fields directly describing the user perceived real-world object are arranged at the beginning of the sequence. However, the arbitrary manner in which one could possibly order field names in a System R view, could result in an awkward sequence and lead to unwanted confusion and misunderstanding in its interpretation. Consider the following fields of a view called Person-Info : Person-name, Post-code, Street-name, House-no, Town, Age, Sex --

Certainly such a view description does not help the end user at all. This example also demonstrates the exposition of semantic links between fields by aggregating the fields of Post-code to Town into a meaningful object called Address. It also means that unnatural sequencing can be better controlled.

Finally to reiterate a point made in earlier chapters, the user is compelled to carry out cross-linkings between base or view tables in order to define a view. This undesirable burden is made worse because connections between fields in same or different tables are not always made explicitly clear to the user. A second more serious setback results from that System R does not support domains. Tables then are connected together as long as the values of the named link fields match. The use of domains is a way of enforcing that the set of values defined by that domain belong to the same category and hence bear a meaningful relationship with each other. In the absence of such a constraint, one could easily and unawarely specify meaningless associations between tables. An example could be where Age of Patient is equal to the number of Lab-tests he or she has undergone.

5.1.2 View definition in SQL

A query is expressed by an SQL 'mapping' operation in the form of a SELECT-FROM-WHERE block. A view is defined by combining the description of the view with the required 'mapping' operation.
The general syntax for defining a view is :

```
DEFINE VIEW view-name ( [field-name ...] )
DEFINE VIEW view-name ( [ view field-name ... ] )
   AS SELECT  [ field-name ...]
      FROM    [ table-name ... ]
      WHERE   [ field-name = 'value' ]
              [ field-name-1 in table-1 = field-name-2 in table-2 ]
```

Notations used are :
... possible repeating items
( ) optional item that can be omitted

The first part of the section investigates the kinds of transformations supported by the language. The second part comments on the general aspects of the language.

5.1.2.1 Mapping capabilities and limitations

Consider initially data item or in this case, field transformations. The list of supported features include:

- o renaming of field - the optional specification denoted that default field, names are used when no new field names are described.
- o computation of a new numeric field by evaluation of an arithmetic expression.
- o built-in summarizing functions, namely, COUNT, AVE, SUM, MAX and MIN.

Others not available are :

- o reformatting of a field type - data types of view fields must be acquired from the underlying base tables o concatenation of data values to form an aggregate field value.

o semantic-motivated mappings - the SQL mapping operation is based on predicates defined on fields of a table. Consequently, it is not possible to project the name of a table as a perceived field value in the view. Due to the uniform modelling of data, relationships between entities in an application are also explicitly represented as tables. It therefore implies a relationship name as well (in the form of a table name), cannot be interpreted as a field value in a view.

SQL, being a relationally complete language [Codd 72], allows a range of powerful set operators that are used to define operations on the extension of a table i.e. all the associated rows or occurrences. As a result, most of the structural transforms are provided, and in the following ways :

a) concatenating transform -

This is executed by a relational natural JOIN predicate. Syntactically, it is represented as the second optional clause in the WHERE statement, as given in Section 4.1.2. For complex views, sub-predicates may be nested to a fixed number of levels. In particular, there is a whole repertoire of powerful commands which can be used to qualify many types of derivation. Some examples are EXIST -- an existential quantifier, ANY, ALL ,etc.

b) subsetting transform -

This is accomodated by the first clause option of the WHERE statement. The selective conditions may use comparison operators =, <>, >, >=, < and <= ; boolean operators AND, OR, and NOT, and parentheses that specify the order of evaluation.

c) merging transform -

This is defined by the set operator UNION on two or more mapping blocks, each of which may use different qualifications.

d) summarizing transform -

This is represented by the GROUP BY operator acting on a named table. The corresponding fields in the SELECT clause i.e. the allowable fields in the derived view, must be single-valued. This means that a field may be the GROUP BY field itself, or a single transformed value as summarized by a built-in function. In addition, it is possible to specify a predicate on the GROUP BY operation by using the HAVING clause.

e) transposing transform -
The hierarchical structuring of repeating values associated with transposing transforms is certainly not meaningful in the context of System R tabular data structures. However, aside from the syntactic structural representation shortcomings, one could still regard the semantic-motivated mapping of the occurrences in its own right; it can be shown that the essence of transposing transforms can be obtained, albeit not in a straightforward fashion, and with limitations. Below are two demonstrations; both are adapted from Section 3.2.2.5.

i) Consider a base table Labtest-A and its derived view, View-A.

```
Labtest-A( P-no, Testno, Result)   View-A( P-no,Result1,Result2,Result3)

          015     10     +ve   =>        015    +ve     -ve      +ve
          015     21     -ve
          015     03     +ve
```

This transformation can be carried out in the following two stages :
1) a series of subsetting transform - this separates out individual occurrences as characterised by Testno, into different views.
2) concatenating the intermediate sub-views from (1).

One possible way of specifying this in SQL is as follows :

```
Stage 1)    DEFINE Subview1( P-no, Result1)
              AS SELECT P-no, Result
                 FROM Labtest-A
                 WHERE Testno = 10
            DEFINE Subview2( P-no, Result2)
              AS SELECT P-no, Result
                 FROM Labtest-A
                 WHERE Testno = 21
            DEFINE Subview3( P-no, Result3)
              AS SELECT P-no, Result
                 FROM Labtest-A
                 WHERE Testno = 03

Stage 2)    DEFINE View-A( P-no, Result1, Result2, Result3)*
              AS SELECT P-no, Result1, Result2, Result3
                 FROM Subview1, Subview2, Subview3
                 WHERE Subview1.P-no = Subview2.P-no
                   AND Subview2.P-no = Subview3.P-no
                                             * the list can be omitted
```

One is immediately conscious that a prior knowledge of what and how many values exist for Testno is required in order to specify the derivation. Note that the intension of this example is similar to the case of repeating - linear transposition.

ii)Now consider a modified base table, Labtest-B and its derived view, View-B.

Labtest-B(P-no,	Result1,	Result2,	Result3)		View-B(P-no,	Result)
015	+ve	-ve	+ve	=>	015	+ve
					015	-ve
					015	+ve

Again, two steps are used to derive the view i.e.,

1) defining subviews for each Result type, this time by excluding the irrelevant field types.
2) merging these subviews together.

The corresponding SQL specification is as follows :

```
DEFINE View-B( P-no, Result)
  AS SELECT P-no, Result1
     FROM Labtest-B
     UNION
     SELECT P-no, Result2
     FROM Labtest-B
     UNION
     SELECT P-no, Result3
     FROM Labtest-B
```

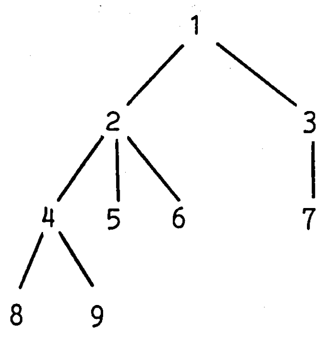
Unfortunately though, the UNION operator automatically removes duplicate rows of the table formed. There is therefore a loss of information, as happens in this case where rows occurring in View-B are just [015 +ve] and [015 -ve]. The third row no longer appears and hence gives the impression that the patient with P-no 015 had only two results, which is not true.

The same snag of having to know specific information a priori to defining the above views applies equally to recursive applications. The latter cases with levels of nesting are even more difficult to handle.

SQL provides a labelling mechanism to reference rows in the same table for the purpose of specifying recursive joins with the table itself. Take the example of the base table Product(Part-no, Comp-no, Qty). The representation of its extension as a recursive hierarchy of occurrences is :

Product(Part-no,	Comp-no,	Qty)
1	2	100
1	3	200
2	4	150
2	5	100
2	6	220
3	7	60
4	8	200
4	9	170

->



Consider the following specification for extracting the immediate components of a part.

```
     DEFINE View-C( Part-no, Comp1-no, Comp2-no)
       AS SELECT R1.Part-no, R1.Comp-no, R2.Comp-no
          FROM Product R1, Product R2
          WHERE R1.Part-no = R2.Part-no
            AND R1.Comp-no < R2.Comp-no
```

R1 and R2 are two arbitrary labels used to specify the rows to be joined under the conditions that the value of Part-no in one row is equal to the value of Part-no in a second row. The other qualification is required to ensure that at most, only one of the pairs (a,b,c), (a,c,b) will be shown.

The rows of the view View-C are :

View-C(Part-no,	Comp1-no,	Comp2-no)
1	2	3
2	4	5
2	4	6
2	5	6
4	8	9

As it appears, the view is not strikingly meaningful; on the contrary, the extension as shown above is not easy to understand because Part-no 2 still has three immediate components and cannot be represented altogether in one single row. This example is purposely chosen to illustrate the awkwardness that can crop up which one does not often find in textbook examples. Obviously, if there were just two components for each part, this problem would not have arisen; the third and fourth row above would not be materialised. Notice also that the base table occurrence where Part-no equals 3 does not feature in the view extension. The reason here being that it has only one component and has no second row with the same Part-no to join with. This is yet another case of missing information.

It can also be seen from the specification that a certain amount of understanding is required for the user to write down the corect join predicates. Second-level components in this example can be obtained by using a join predicate of 'R1.Comp-no = R2.Part-no' in a similar way to that just shown. However, in general any further levels must be nested accordingly to the number required. This means that there can only be a fixed limit of iterations. In other words, it is not possible to extract for each part in the example hierarchy, all the related sets of component parts at every level of the hierarchy, without knowing how many levels exist for each.

So far the single table recursive join mechanism has been used to illustrate the one shortcoming above. It is equally important though, to appreciate its usefulness when applied to non-recursive applications. The operation represents a powerful facility for establishing a relationship between rows (occurrences) of the same table (entity) type. This is analogous to a semantic-motivated mapping. The example below, borrowed from Date [81], aptly depicts this.

Given a base table for Supplier S(S-no, S-name, Status, City) and the SELECT statement,

```
SELECT First.S-no, Second.S-no
FROM S First, S Second
WHERE First.City = Second.City
  AND First.S-no < Second.S-no
```

The meaning of a view defined using this SELECT specification will be 'Supplier-Pairs who can be located in the same city'.

Just to recap, several crucial points emerge from the rather long discussion on transposing transforms.

- o There is a limited form of two-way semantic-motivated mapping of data value to attribute type; any definition also requires a fairly substantial understanding and pre-knowledge of certain underlying data values on the part of the user.

- o Recursive structure mappings is restricted to a fixed depth of iteration decided a priori by the user.

o The abstraction of a relationship (type) between the same entity type occurrences into a user-perceived view is supported.

The next section now looks at some other less specific issues.

5.1.2.2 General criticisms

There are two noteworthy ones here.

i) Awkward formulation of queries.

ii) Incorrect data materialisation in the sense of not meeting user's intentions, from syntactically faultless specifications.

Notice the use of the term 'queries' which is used presently to convey the context of view definition. Studies on the user-friendliness of relational query languages [Shneiderman 78, Reisner 81], have in fact revealed that SQL's 'sugared' syntax is rated very favourably, just after QBE(Query-By-Example; see Zloof 77). Indeed, in most cases of averagely-simple mappings, it is not too difficult. However, with most complex mappings, the nesting of sub-queries can become very cumbersome; this is especially true of recursive applications. The ripple effect of this is partly described by the second drawback listed above.

When forced to embed queries to several levels, which requires a careful and thorough understanding of the underlying data structures and their connections, one is prone to making logical slips. Very often in such circumstances, the syntax of the specification is not violated. Consequently, the slips go undetected, resulting in data whose semantics is inconsistent with user expectations, and which the user is not aware of. There is also no proper means of checking out such expectations. Hence, one cannot properly validate the retrieved data, except by knowing beforehand the underlying data occurrences so as to be able to work out manually what the expected data values should be. Such an ad-hoc solution, with its unreasonable and unrealistic assumption of pre-knowledge, is not at all practical and is only feasible with very small databases (such as those used for course presentations at universities). The absence of effective semantic validations is by no means confined to System R; it is probably true of all systems. In System R, this factor, coupled with a number of others, bear a serious implication on the semantic integrity of its view mechanism.

5.1.3 Flexible view definition : an implication

There is no concept of a centralised role of a DBA in System R as in say, a Codasyl-based DBMS. Individual users define their own views as long as they have the appropriate privileges granted. The external 'schema' here consists of a collection of base tables and derived view tables. The user can specify new views either on the underlying base tables or on existing view types. Given that there is no guarantee of the semantic interpretation of views, semantic inconsistencies can easily proliferate. Further views derived from inconsistent ones can only inherit the consequences. The preceding section has already underlined one possible source of semantic inconsistency which is related to the language and query specification. A second possibility is due to the way that integrity constraints are handled optionally, and significantly that domains are not supported. Unintended meaningless derivations more often than not will pass unnoticed. Finally, the low level of semantic expression in the data representations that can be perceived by users, may also contribute indirectly, as a result of sheer misinterpretation.

5.1.4 Conclusions

The mapping facilities as provided by SQL are relatively powerful. This is not surprising given the relational completeness of SQL. Semantic-motivated mappings however, remain an important gap to be bridged.

It is clear that there are major inherent inadequacies of a tabular view structure. Proliferation of semantic inconsistencies is undoubtedly, a constant threat.

In comparing a System R view with the declared view characteristics of this thesis, one can readily conclude from the above studies that a System R view does not conform to a unit view structure which requires no cross-linking of data structures by the user. On the other hand, direct data access, in terms of providing processed data is very good.

If a query is modified into a view, it is likely to be due to the following reasons :

i) an existing view table provides easy access to data by other users.

ii) the query is a repetitive transaction and hence it is useful to keep the definition.

iii) as a means of access control - by extracting out only certain types of data in a view, selective data can be protected.

iv) as an aid to schema structuring - consider the restructuring of base table by replacing a given base table by two of its projections such that a join of the projections will produce the original table. An important requirement of schema changes caused by the restructuring is that users should not be affected. By constructing the original table as a view table defined over two projections, the user is immuned from the underlying structure modifications. This though, is only true as far as data retrieval is concerned.

The conclusions that can be drawn are :

o the view mechanism provides logical data independence to users when schema restructuring is carried out.

o the view mechanism of System R probably stands out more prominently as an access control mechanism. This claim can be backed by the considerable amount of research efforts devoted to its view authorization and access control mechanism [Chamberlain et al 75, Griffiths and Wade 76, Selinger 80, Fagin 78, Stonebraker and Wong 74]. (The last reference though not directly related to System R, has its relevance in that it addresses access control issues in relational views defined by means of query modification).

5.2 The Codasyl EUFC Forms-Oriented Approach

The Codasyl EUFC (End User Facilities Committee) proposals aim specifically to support a general environment where end users handle forms - office forms, worksheets, memoranda, reports etc., in their daily work activities. The proposals are still under development and the studies reported in the first version of its JOD (Journal of Development) [Codasyl EUFC 83] on which this evaluation will be based, concern the derivation of end-user view objects only from a database definable by the Codasyl Data Description Language [Codasyl 78]. Again, aspects other than those related to view provision in the JOD, will not be addressed here.

In the rest of the discussion , a 'form' is defined to be a rectangular object which is the primary unit of data perceived and manipulated by the end user. A group of forms may be collected into a higher level object called a file. Forms and files can be of two types :

o base perception forms and files represent user views of the underlying database and are defined by a DBA using a data description language called EUF DDL.
o base user objects are defined by the end users themselves using an OMIL (Object Manipulation Interface Language) -- they are only relevant as far as the OMIL operations are concerned, which allow end users to create derived perception objects upon existing base types.

A form has a hierarchical structure; it can be regarded as a special class of end-user view - this will be discussed in the next section of 5.2.1. Section 5.2.2 examines the mechanism adopted for database-form mapping. Its translation into the EUF DDL syntax and the extent to which transformations are supported will be described in Section 5.2.3. Section 5.2.4 briefly concludes the findings.

5.2.1 The form as a user-view

The use of a form as a view represents a very good example of abstracting away from underlying database structures to a real-world object with which the user is familiar and identifies readily. Its usefulness in encapsulating rich semantics is expressed in terms of its :

- o concept
- o basic hierarchical structure
- o positional layout of overall data that enhances the semantics

The concept of a form is an idea (of a thing) that is very well established in our human minds. A form regardless of how its contents are obtained is still to the end user, just a containment of some specific information -- he or she can be completely divorced from any knowledge of the database. Furthermore, the user will not have to learn any newly-coined term for an object which although represents the same concept, will nevertheless take some time to adjust to.

The hierarchical organisation of data is the chief aspect that merits discussion. The basic structuring construct for a form is a group. A **group** is a complex of one or more elementary data items, referred here to as group items, and other possible nested groups. The notion of a base entity introduced in Chapter 1 is met by the equivalent of a root (group) for a form. Groups other than the root group can contain repeating values.

The advantages of hierarchical data representations in general, have already been previously mentioned and will not be dwelled on further. Two EUFC specific limitations related to the specification of a form structure will be considered later.

Finally the provision of geometric layout facilities is undoubtedly an additional means of improving user's semantic perception, thus reinforcing the effectiveness of the form as a user view. This however, is a secondary concern and will not be part of the study here.

5.2.2 A non-direct mapping mechanism

The mapping process consists of three distinct logical stages as depicted below :

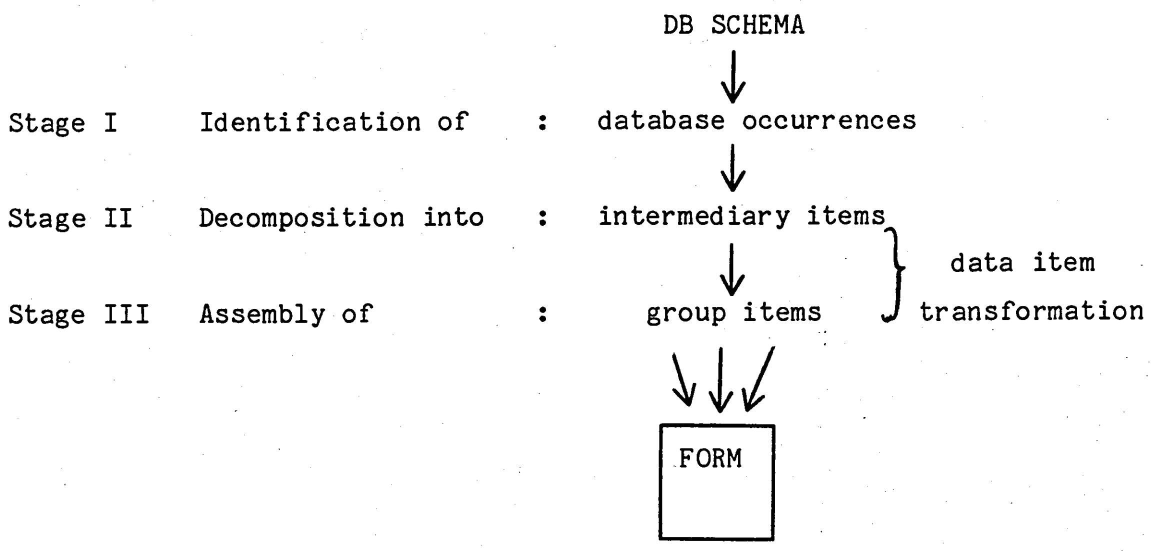


fig. 5.1 Mapping of an EUF FORM

Each stage is represented independently by an EUF DDL section. The key element of this non-direct derivation is the pool of intermediary items. These intermediary items are database items belonging to the subset of (Codasyl) record occurrences selected. Though it is required that every form item must have its intermediary correspondence, not every intermediary item must appear on the form because :

- o it may be used in the transformation of a different group item,

- o it may be used as a qualifying attribute for the exclusion of irrelevant occurrences of some other group item types or group types which satisfy the conditions (see Item/Group Limiter Section later). Group items are therefore either transformed from intermediary items or map direct from them. All group items must necessarily be contained in a group construct for the purpose of structuring the ultimate form.

One possible advantage in thus breaking up the process is that it provides a kind of canonical source of intermediary items which is independent of the underlying model and database structures. One could conceive the derivation of the intermediary items from more than one database occurrence type i.e. relational or hierarchical besides the presently-supported Codasyl network database, which may require different specification for identifying the correct set of data occurrences.

There are however, a number of disadvantages. These are as follows :

o The most severe criticism is that the canonical pool of items is at too low a level. The semantic associations between database records i.e. whole collection of database items, are broken down and there is no mechanism to preserve them in the re-assembling into semantic units for a form structure. For example, as long as the item name used in a data item transformation algorithm exists in the intermediary list, the group item mapping will succeed even if it may not be semantically correct. The arbitrary structuring of the group items into groups establishes a relationship between these groups which may not reflect the true underlying semantics. Again as in System R, there is no formal way of validating the form contents.

o The necessary containment of group items in a group means that in the case of a single item group, such a step is not meaningful - it is a mere duplication of names. The duplication is particularly true of the additional naming of a root group which in essence refers to the entire form itself. The root group name hence can be equally represented by the form name. In view of the fact that a view (as described in Section 1.2) has the same basic hierarchical structure, this point will be carefully noted in the subsequent proposal of a view model in Chapter 6.

o In instances when it is required to map a whole database record occurrence into a semantic group in a form, the decomposition stage proves to be unnecessary; additional work is required which increases the probability of incurring errors. It is also here in the reconstruction stage that the original essential semantics can be contaminated.

5.2.3 Defining perception objects in EUF DDL (Data Description Language)

Not surprisingly, the syntax of the EUF DDL resembles other Codasyl specifications and employs entries, sections and clauses. The language consists of two main parts :

o the Form Entry which specifies the definition of a base perception form and its derivation.

o the File Entry which specifies the definition of a base perception file containing selected forms.

Form Entry is further divided into the following ordered Sections :

i)Source Section - this simply states the name of the database schema to be accessed.

ii)Name Section - this specifies the name of the form.

iii)Item Section - this defines the list of intermediary items and the specification of any desired transformation of form items. It represents the decomposition stage of the process.

iv)Structure Section - this describes the hierarchical structure of the form in terms of the component groups and their corresponding group items. It represents the assembling stage.

v)Data Section - this defines how the underlying database occurrences are to be selected. It represents the identification stage.

vi)Item/Group Limiter Section - this determines the conditions for limiting number of occurrences of a group item type or group type to be displayed in the final form.

vii)Geometry Section - this specifies the visual appearance of the form layout.

viii)Description Section - this gives a text description of the form entry; it provides a full explanation of the form semantics to the user -it is good recognition of the need for such an important facility.

The File Entry has the corresponding sections of :

i) Name Section - this names the file.

ii) Form Section - this defines the form type of the file.

iii) Limiter Section - this specifies the criteria for excluding form

iii) Limiter Section - this specifies the criteria for excluding form instances in the file.
vi) Description Section - this is a text description of the file entry.

Only two Form Entry Sections, namely Item and Data, will be further discussed in detail.

5.2.3.1 Mapping capabilities and limitations

The types of item transformations supported are first considered. These are :

o renaming of data item.
o reformatting of item type - this is expressed by the familiar COBOL PIC clause.
o creation of a new numeric item using an arithmetic expression.
o provision of summarizing functions - the functions here are not yet built in as in System R. Instead, they are specified as pieces of named procedures by the DBA. It is noted though that codes can be generally written for other more complex transformation algorithms besides summarizing functions. This flexibility can be used to generate fairly powerful mapping but it also implies that external semantics may be introduced unintentionally in syntactically-correct procedures.
o transformation using coded table rules - this can be achieved by representing the translation rules in an external procedure.

Two classes missing from the above list are :
o derivation of an aggregate item by concatenating the values of elementary source items into a single item value. Notice that this is not the same as combining several items together in a group where the group items are made explicit.

o semantic-motivated mappings - all group item mappings are from data items only. There is no provision for deriving form contents from database record names or Codasyl set names (relationship names).

Given that the underlying database is a network, the major structure mapping supported by the EUF DDL is expectedly, that of :

a) concatenating transform -

The Data Section contains a number of traversal commands for navigating through the network structures. The origin node of a concatenation is identified using the SEARCH clause. The clauses THEN and FROM are means of specifying what database record types are to be visited and the set type to be used. Two interesting features are to be examined in more detail :

i)<u>recursive path definition specified by the EXAMINE clause</u> -

The clause allows the recursive looping of a pair of Codasyl owner-member record types that are related by more than one set. The syntax is as follows :

```
EXAMINE RELATIONSHIP record-name-1
   THEN record-name-2
   BY set-name-1
   OWNER record-name-1
      BY set-name-2
      GIVING item-name-1 [item-name-2]...
[UP TO integer-1 TIME [S] ]
```

Note that record-name-1 and record-name-2 must be connected in an owner-member direction. Record-name-1 must be previously stated in the clause directly preceding the EXAMINE clause. The UP TO phrase specifies the terminating condition for the recursive looping. INTEGER-1 indicates the maximum number of times the pair of sets (set-name-1,set-name-2) will be traversed between the initial retrieval of record-name-1 and the subsequent retrieval of record-name-1. This is taken to mean the number of levels of the recursive hierarchy to be extracted. If the phrase is omitted, no maximum condition is assumed. This implies that the mapping is not necessarily restricted to a fixed number of levels that must be explicitly defined.

One contention remains with the specification. Item-name-1 and item-name-2 must be derived from schema items from record-name-1. This in a way, limits the expressiveness of the operation to single-loop hierarchies -- in Codasyl structures that the EUFC is based on, the same record type cannot be both owner and member of the same set type, hence disallowing loop structures. To overcome this, a second link record type can be introduced with two set types connecting the original record type and new link type. In this case, the link record is not required to contain any data items at all.

Consequently, the constraint on the item source is not significant if the link record (represented by record-name-2) is in fact empty. This, however is not true with complex recursive loops. Here the link record does play a meaningful role and may contain useful information. For instance, in the part assembly example, record-name-1 may be replaced by PART with data items Part-number, Part-name and record-name-2 replaced by COMPONENT with data items Component-number and Quantity-used. With the specification as such, information about the number of units of component used cannot be derived.

ii) <u>concatenation based on item value matching specified by the LOOKUP clause</u> -

The LOOKUP clause allows a relationship to be defined between (occurrences of) two record types given that the matching predicates are satisfied. It is equivalent to a relational JOIN operation. This is a useful facility for new relationship types that can be derived independently of set relationships.

Support for other classes of structural transformations is as follows :

b) subsetting transform -

Instead of deriving a new form object (type), an entirely different object type, a file is created. The introduction of the file concept is yet another attempt to employ objects familiar to end users -- a file actually conveys the meaning of a whole collection of things.

The specification is contained in the File Entry. The selection condition is expressed in quite different terms. Called the limiting criteria, it specifies the predicates used to filter out a subset of irrelevant form instances. To demonstrate this, suppose a Late-Order-File is to be constructed containing form instances where form item Order-date > 30-1-84. The limiting criteria in this case will be to exclude form instances when Order-date <= 30-1-84. This somehow seems to a rather unnatural way of stating the requirement because one has to twist the criteria the other way round.

c) merging and summarizing transforms -

As it stands, these are not featured by the EUF DDL. The reason can be attributed to the traditionally-lacking support for relational set operations in Codasyl-typed DBMS where record-at-a-time traversals are more common. It is however becoming increasingly recognised that such powerful set operations are needed to extend the flexibility of Codasyl-based systems. One witness to this is CQLF (Codasyl Query Language Flat) [Manola and Pirotte 81,82] -- the SQL-like language is an attempt to enhance the retrieval power from Codasyl-based databases by supporting a range of relational set operators.

It is very interesting to note though, that external to the EUF DDL, there exists a MERGE operation in the OMIL which the end user can use to define a derived user file from a number of source files. The MERGE operator has very similar semantics to that defined for a merging transform, except that the source files must be ordered in the same way. Consequently, it is not strictly true that merging transform is not at all possible. The capability does exist at the end-user level. It must be pointed out that although the results achieved may be the same, the implications are different. By placing the task of specifying the operation in the hands of the end user, the end user has to explicitly carry out the merging. If the facility is included in the lower realms of the EUF DDL, the end user can be presented with both the transformed entity and the individual entities in their original 'form', depending on choice. The foregoing statements are not meant to imply that the availability of such facilities should be removed from the end users, but rather that if the objective is to provide direct data access to end users, then the feature should be equally included at the database-mapping level.

d) transposing transform -

It can be envisaged that the transposition from linear to repeating structures and vice versa, is possible by selecting the required correct item occurrences. The limitation is as in System R, where explicit values must be known beforehand. This last criticism is directed at the absence of means to extract information that may be carried in a Codasyl set construct and set ordering. The position of a record in a set may contain a useful piece of information, as in an example given by Kay [75] -- in a PARENTS set, the position of the set member may be used to identify the two parents (which otherwise can be identified by a data item SEX in the record). Another example of desirable information is the number of members that are associated with a given set.

The same problem of incorrect data materialisation prevails too with the EUF DDL specifications. Here again there is no way to find out if the semantics is not as expected. As already pointed out in Section 5.2.2, the intermediary item mapping process itself is inherently very prone to semantic consistencies. On the other hand, there is perhaps better check on the propagation of such inconsistencies in a DBA-controlled environment since end users are not allowed to manipulate database objects directly and freely. As such, the possibilities of committing errors from a lack of understanding are reduced.

5.2.4 Conclusions

The Codasyl EUFC proposals represent an encouraging step towards the direction of providing real-world views of data from databases. Its mapping facilities can certainly be upgraded by the inclusion of :

- o set operations which support structural transforms other than concatenating and subsetting,
- o semantic-motivated mappings,
- o facilities to materialise data inherently held in Codasyl sets.

The major drawback is the loose semantic preservation that can be maintained by the non-direct mapping mechanism. Nevertheless, it has the potential of supporting a more general environment of heterogeneous databases and commercial files.

Chapter 6 THE PERCEIVED RECORD VIEW MODEL

Section 1.2 in the introductory chapter has described the desirable properties of a high-level, end-user view. This chapter now considers these specifically and defines a view data model that :

o supports such a view, hereby formally called a **perceived record,**

o can be easily built on top of a database conceptual EAR model,

o can provide the basis for building a non-DBMS specific, easy-to-use query language -- the ease-of-use requires that the query language defines on a minimal set of data constructs.

The following section first clarifies a number of terms and associated concepts. Section 6.2 then examines the basic data construct used to model a perceived record view - the component construct - and the roles it play. Alternatives for specifying the data model in terms of the component construct are investigated in Section 6.3. The last section, Section 6.4, looks at the constraint of uniqueness.

6.1 Some concepts and definitions

Two notions must be clearly distinguished.

The perceived record data model describes formally the rules of how its data objects relate to one another, and the semantics of these data objects.

The perceived record view, or simply known as the perceived record, is a unit data structure that is defined by the model. It represents :

A named collection of data values that contains all the information that the user is interested to query about.

These data values may be constructed from a number of database entity occurrences which may not necessarily be of the same type. The definition carefully avoids implication of any specific form of structure; this maintains the flexibility as a perceived record may be both :

o a flat view where all the data values are single valued.

o a structured view where it consists of a number of embedded groups with data values that may repeat.

Fundamental to a perceived record lies the concept of a base entity. As noted in Section 1.2.1, a base entity corresponds to the user-perceived real-world object; the essence of the perceived record semantics is described in terms of the base entity type. Such a base entity type must be mappable from an existing database entity type. In particular, one occurrence of a perceived record type is represented by one instance of the base entity type that specifies the user's perspective. This effectively means that a perceived record occurrence captures the required semantic information about one precise object instance that the end user perceives in the real-world.

All data values in a perceived record belong to the corrresponding perceived record items. A perceived record item (abbreviated to pr-item) is the smallest unit of data in a perceived record. It is basically similar to a data item.

The basic structuring construct is the component. This is defined to be a named group of pr-items which possess a common association. The association can be implicit or acquired. Implicit association refers to the relationship that exists naturally between attributes which together describe the characteristics of an entity. For instance the attributes of D-name and D-phone-name are held implicitly together as one aggregate unit conveyed to the end user as the Doctor entity.

Acquired association is more complicated. It defines an explicitly derived relationship between two or more pr-items brought together by some form of structural transformation on the same or different data object types.

6.2 Roles of the component construct

Before one can start building a data model, one needs to fully understand the purposes of the data constructs used. A component has four main purposes :

i) At the occurrence level - structuring

The component construct provides a facility for handling repeating values of pr-items. This capability is one of the chief view requirements identified earlier on in Section 1.2. It needs to be reminded though that data values in a component do not necessarily repeat.

ii) At the data type level - naming

Obviously the identification of a list of pr-items by a meaningful component name projects out more clearly to the end user, the intent of the data. This applies usefully when a component is mapped directly from one whole database entity; the name of the database entity can be retained as the component name if so wished. Such cases illustrate components bearing an implicit association.

iii) Acquiring new semantics

A component consisting of pr-items related by an acquired association represents a newly derived semantic data object. In this way, a component may in fact be considered as a view object. This will have certain implications in terms of defining the perceived record model.

It is interesting to note here that the 'group' construct featured in the Codasyl EUFC forms approach, does not support such a role. Subsequently, the functionality is very much reduced to a merely syntactic one of 'putting' group items together.

iv) Nesting of components

A component can be used to embed other sub-components, thus providing the potential to model complex structured views. The diagram below illustrates the nested component structure. Each component has zero, one or more sub-components, expressed by the relationship 'has'.

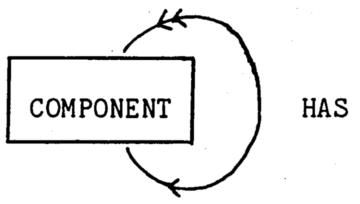


fig. 6.1 Nested structure of the component construct

Given the above purposes of the component construct, the following section proceeds to find the most efficient way of specifying the complete data model, both in terms of usability to the end user and ease of mapping.

6.3 Alternative ways of structuring

The design of the perceived record view model is based on entity modelling, that is, identifying the entities involved, their roles and their constituent attributes. The entities will be in third normal form (3NF). A single database environment is assumed throughout. The data object representing the perceived view thus can be sufficiently uniquely identified by the name of the data object name alone; otherwise, a composite key comprising of the database name is required.

There are three different ways of expressing a perceived record data model :

- o as a special case of the component construct
- o as a hierarchical structure
- o as a variable typed structure

Each of these alternatives is described in turn.

6.3.1 The component as a perceived record

Recall in the previous section that a component can assume the role of a view. The proposal here exploits this fact by modelling the perceived record view as a special class of component which is used for retrieval purposes. The component construct still retains its nested structure, but now plays two distinct roles :

- o as a perceived record view entity.
- o as a sub-component.

The data model can be represented diagramatically as follows :

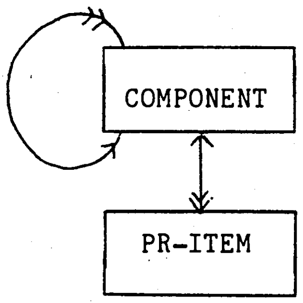


fig. 6.2 The Component Entity as a special class of Perceived Record

The following statements describe the above diagram.

- o A component name (c-name) is unique within a database.
- o A perceived record item name (pr-itemname) is unique within a component.
- o A component type contains n pr-item types and m immediate i.e. at the next level component types where

  n is an integer between 1 and a finite number N,

  m is an integer between 0 and a finite number M.

In this design, a component featured as a perceived record, becomes a data object that can be manipulated. It is no longer just a semantic and structuring construct. Queries built on this model are therefore based on components which in turn reference pr-items. The basic operation that can be defined on a component is 'GET' where this can be interpreted as retrieving data values corresponding to the component.

One far-reaching implication arises from this. Every existing component described in the external schema must have only one unique name and distinct role. This is necessary to ensure that queries defined in terms of the component names are unambiguous. The parallel of such a unique naming constraint occurs in universal schemes (refer back to

Section 2.3.2) whereby all attributes must bear one unique name and meaning for the purpose of attribute-based queries. The same criticism there is true here - a long list of component names must be carefully maintained at all times. This list can easily proliferate especially in the case of a large database applications. Consequently, this model is not a practical solution.

The above drawback can be resolved by identifying and modelling the two roles explicitly into two separate data objects. The perceived record retrieval role can be abstracted into a higher level object. The next model explores this mechanism and introduces an additional data object - the perceived record entity.

6.3.2 A hierarchical structure

This model features a perceived record entity 'sitting' on top of the previous one. The resultant data structure is hierarchical - a perceived record entity is expressed in terms of components which in turn break down into pr-items. The diagram below illustrates the representation. PR abbreviates Perceived Record.

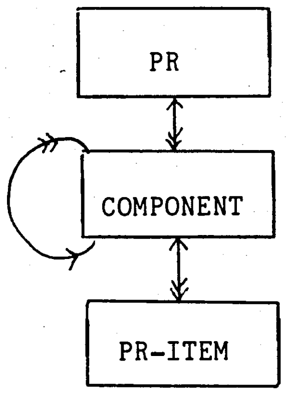


fig. 6.3 Hierarchical representation of perceived record model

The following statements are true of this model.

- o A perceived record name (pr-name) is unique within a database.
- o A c-name is unique within a perceived record.
- o A pr-itemname is unique within a perceived record.

o A perceived record type contains n immediate component types where
    n is an integer between 1 and a finit~~i~~e number N.

o A component type contains p pr-item types and m immediate component types where
    p is an integer between 0 and a finite number P,
    m is an integer between 0 and a finite number M.

The component entity now becomes strictly a sub-structure within a perceived record entity. The second statement above reflects its subordinate role. As it is, it cannot be treated and manipulated as a data object independently of the perceived record entity that it belongs to. The unit of data referenced and manipulated by the query language is now replaced by the perceived record entity.

In overcoming the limitation of the first model, this solution presents its own disadvantages as well. The uniform hierarchical structure forces a rigidity on an end users' perception of a perceived record; all pr-items must be contained in component constructs and hence be referenced via component names. The base entity of a perceived record view is best expressed ~~by~~ in terms of pr-items belonging directly to the perceived record entity, without the need of a duplicated level of component structuring and naming. The representation of the view in Chapter 1 (page 8) based on this model (view semantics remaining unchanged) is as follows :

```
PATIENT( Pat-details(P-no, P-name, Appmt( Appmt-date, Doctor( D-name,
                        D-phone-no ) ), Test( Type-of-test, Result ) ) )
```

As noted then, the base entity of the perceived record PATIENT is the database entity Patient. This particular piece of semantics is already captured by the name of the perceived record view. The additional specification of a component structure and hence the appearance of the component name of Pat-details is redundant. Moreover, it may cause confusion to an end user using the perceived record. One could argue though that the component name, equivalently expressed by the perceived record name can be suppressed from the display of a perceived record type to the end user. This effectively is the approach adopted by the Codasyl EUFC proposals. The visual rearrangement hides the fact that the underlying hierarchical structuring of 'groups' is inflexible. The inherent rigidity however remains untackled.

The final model below attempts to improve on this design with a more flexible structure.

6.3.3 A variable-typed structure

The approach here removes the strict hierarchical relationship in the above design by expressing a perceived record entity as a combination of both components and pr-items directly. It allows a perceived record to consist only of pr-items, and if required, the use of components to structure it. The data model is represented as follows :

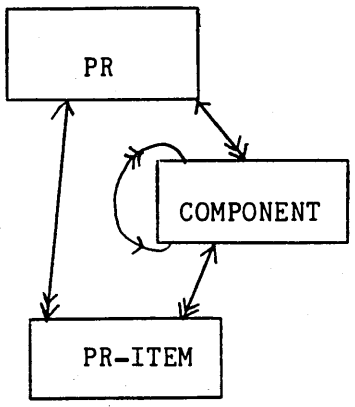


fig. 6.4 Variable-typed representation of the perceived record model

The following statements are true.

o A pr-name is unique within a database.
o A c-name is unique within a perceived record.
o A pr-itemname is unique within a perceived record.
o A perceived record type contains n pr-item types and m immediate component types where n and m are as before.
o A component type contains p pr-item types and q component types where
  p is an integer between 1 and a finite number P,
  q is an integer between 0 and a finite number Q.

Note that in the last statement, when p=1 and q=0, the component consists of just one pr-item. Since a perceived record entity can be described in terms of pr-items directly, this additional level of referencing via a component is not particularly meaningful. This contrasts with the second model discussed above, where such a structure will be absolutely valid as enforced by the inherent hierarchical constraint.

The variable-typed representation has two attractive properties.

i) Flexible structuring -

Following from the above definitions of the model, two types of perceived record structure can be distinguished.

A simple perceived record comprises just a list of pr-items that are associated to the base entity; it contains no components. This can be regarded as representing a flat, non-hierarchical view.

A complex perceived record consists of a number of pr-items related directly to the base entity, and one or more components. It represents a nested structured view.

ii) Common derivation -

The last two assertions made at the introduction of this model state a structural equivalence between a perceived record entity and a component entity. Both structures are described by the same set of rules. In other words, a perceived record entity and component entity can be derived via a similar route. This is hardly surprising as the perceived record entity evolves from abstracting one of the dual roles of the component construct. The previous model has failed to capture this point because a perceived record must always be defined solely in units of component at one level, and pr-items at a lower level.

A simple pr, consequently, can be regarded as a component comprising of only pr-items and no other components. It follows then that a complex pr can be thought of as a homogeneous combination of components. Clearly, this model offers a great potential for developing a powerful and flexible mapping mechanism based on one uniform and symmetric class of data object. This property influences very significantly the way that the mapping mechanism will be formulated.

The perceived record entity represents the only unit of data that the query language, and hence the end user manipulates. Just as in the second model, queries only reference the component construct contained within a specified perceived record entity. The kinds of query language operators defined on the perceived record entity will be simple retrieval commands, such as the pseudo 'GET' operation described earlier. Higher level operators that one accustoms with many data manipulation languages and especially relational query languages (for example, the JOIN or GROUP BY operator) will not be considered. This accords with one of the initial motivations of this thesis to propose a view model that can provide a basis to build a uniform DBMS-independent query language which requires no excessive manipulative operations for the purpose of data retrievals.

To complete the full description of the data model in fig. 6.4, the following specifies the properties of the perceived record entity, component entity and pr-item entity of the model, as attributes of these objects. Identifying attributes are underlined. ... indicates additional attributes that define the mapping rules; these will be included in the next chapter.

Perceived Record(pr-name, ...)
Component (c-name, pr-name, pc-name, ...)
Pr-item (pr-itemname, pr-name, c-name, pri-role, pri-format, ...)

Some explanatory notes :

o Component has a composite key since a component is unique within a perceived record.

o Pc-name abbreviates parent-component name.
Referring back to the component structure diagram (fig. 6.1), each component can contain components of its own i.e. sub-components. Equivalently, each component has one parent-component name at one immediate level. The attribute parent-component name defines the relationship 'has'; it is functionally dependent on c-name and pr-name. Where a component belongs immediately to a perceived record entity, i.e. it does not belong to a component, the corresponding pr-name has a null value.

o The 1:n relationship between component and pr-item is represented by putting c-name as an attribute in the pr-item entity. Each pr-item is sufficiently identified by the composite key of pr-itemname and pr-name uniquely. C-name does not form part of the identifier of pr-item entity. It will be incorrect to do so, for where a pr-item belongs directly to the perceived record entity, c-name will have a null value; it hence cannot be part of a key.

o Pri-role abbreviates pritem-role. This denotes whether a pr-item is an identifying key attribute or non-key attribute for its associated perceived record type. Pri-role will have values 'K' (Key) or 'NK' (Non-Key). (See Section 6.4 on key constraints).

o Pri-format (pritem-format) describes how the data value of the pr-item type is to be represented in an occurrence of the perceived record. At the most basic level, pri-format takes one of these values : INT (integer), STR (string), REAL (real number).

One more aspect of the data model needs to be described in detail, and that is the issue of data constraints.

6.4 Constraint of uniqueness

The discussion here specifically concerns key constraints only. Other forms of constraints, especially consistency ones that must be maintained during the mapping process will not be dealt here. Instead they will be considered in the development of a mapping mechanism.

A fundamental requirement of the perceived record concept states that each perceived record occurrence represents only one distinct real-world object instance that the user can unambiguously identify. It means that all perceived record occurrences for a given perceived record type must necessarily be uniquely derivable from the database.

Uniqueness is established by one pr-item or a combination of pr-items which possess unique data values for each occurrence of a given perceived record type. Pr-items bearing this unique identification property are known as perceived record keys (prkeys). In order to fulfil this integrity constraint, a perceived record must have prkeys. This fact has been modelled by the inclusion of the pri-role attribute for the pr-item entity in the previous section.

In general, identifying database attributes of the database entity corresponding to the base entity form the necessary prkeys of the mapped perceived record type. Cases may arise though, that composite prkeys in fact derive from different database entity types. An example of this will be demonstrated in the next chapter.

It must be stressed that this constraint must not be imposed on end users such that they have to be actually aware of prkeys in order to retrieve perceived record occurrences; otherwise the whole purpose of the perceived record approach is defeated. Only the DBA or EUA who specifies the mapping of a perceived record type needs to understand and know what prkeys are to be used.

Chapter 7 A GENERALISED CANONICAL TRANSFORM MODEL

This chapter aims to develop a formal method of specifying the correspondences between the data objects in the conceptual schema and those in the perceived record model. The proposed mechanism makes use of an intermediate pivot structure. Section 7.1 first examines existing mapping techniques and explains the reasons for adopting an intermediate approach. Section 7.2 presents an overall view of the underlying principles. The actual design of the transform model is conducted in three main parts in Section 7.3. The chapter concludes by appraising the proposal.

7.1 The Rationale

7.1.1 Methods of mapping

It is useful first to consider methods currently used to achieve view transformations from database structures. The context of database structures is stressed in order to distinguish the related issue of mapping views based on predefined view definitions. This is usually provided as an additional enhancement to the basic mapping capabilities of a system -- it allows appropriate mapping definitions to be reused without having to duplicate efforts, thus reducing the probability of incurring errors.

Two common mapping techniques are as follows:

a) Direct correspondence of view structures from database structures -- Relational query language systems usually employ this strategy. As already noted, System R is one example, and so is Ingres. This method has the advantage of being a neat, one-step procedure. It however suffers from the drawback that mappings are restricted to a specific database model - the mapping definition language must be defined on a set of data structures and in this case, those of the database model. Hence, unless this set of structures is generalised to support different specific logical data models, view mappings will be constrained to the only one data model that the database is built upon.

b) Use of intermediate data objects --

This strategy involves two steps : first, the translation of the database structures to some intermediate form, and second, the construction of these intermediaries into the required view structures. The intermediaries may conform to a data model e.g. a functional data model as in MULTIBASE, or may be an unstructured collection of elementary data types that can be represented by some kind of an end user data model at a later stage. The Codasyl EUFC mechanism clearly illustrates this latter approach.

This method entails more efforts both in coding the rules for the two steps, and in maintaining the consistency of the mapping throughout. Consequently, extra consistency checks are required to validate the correspondence between the intermediaries and database schema objects. The considerable amount of validations can adversely affect the system's performance and efficiency.

On the other hand though, it does not have the disadvantage inherent in the former direct approach -- intermediate data objects provide a means of supporting generalised mappings by accomodating different data models into common structures. The flexibility of derivations depends very much on the choice of the intermediary data structures, as for example the kinds of inherent constraints that may be imposed by a particular data model. In a hierarchical model for instance, 'all children' objects must have one 'parent' and hence the 'parent object' must be referenced for every derivation. (For further discussion on inherent constraints, refer to Brodie [78], Tsichritzis and Lochovsky [82]).

7.1.2 A canonical approach

The above discussion helps to clarify the possible avenues together with their merits and demerits that can be considered for perceived record mappings. It is important to emphasise that the source structures for mapping perceived records considered in this work, are not specific to a given logical data model of DBMS. As such, the factor of supporting (or not supporting) generalised mappings in the two approaches does not influence the choice of technique. Rather, the chief reason for adopting the second intermediate strategy in this case, arises from the non-monolithic structure of a perceived record. A perceived record view may be a simple flat structure or a complex one with embedded repeating groups. The task of developing general algorithms for the direct structural correspondence in terms of database objects is non-trivial. A way out will be to tackle the problem in a modular fashion by breaking the process down into a number of functional steps, with common mapping characteristics described in simple intermediate mapping objects.

We further propose that the mapping be via a **canonical transform model** built on simple data structures. There are two main reasons. First of all, the use of elementary unstructured data items as occurs with the EUFC approach leads readily to semantic inconsistencies. Using larger structured units of data helps to alleviate the problem because a data structure constrains that the associations between the constituent data items be semantically meaningful. The problem of maintaining semantic consistency in constructing a view from the intermediaries becomes a more manageable task of identifying associations between blocks of data (as opposed to individual item), with a reduced probability of errors.

The second reason lies within the perceived record structure itself. As explained in Chapter 5, the structure of a perceived record can be expressed homogeneously in terms of the component construct. It is appropriate to define a generalised flat object, hereby referred to as the **canonical mapping object**, in the transform model that corresponds to the component construct. Many different types of perceived record can then be derived using a common data structure.

The canonical mapping object therefore forms the key feature of the canonical transform model. Its roles are :

o to describe the properties of a component (type) i.e. the rules that are applied to derive the component from the database, any transformation operations that define a structural reorganisation of its source types, and the data items of the component.

o to facilitate the derivation of a perceived record view from one uniform type of data object. The canonical mapping object can be regarded as the basic building block for a perceived record.

The overall objective of the canonical transform model, in summary, is to provide a simple, but yet general means of capturing the mapping information necessary to derive a perceived record view from the database. A very important requirement of the mapping is the semantic integrity of the occurrences of a given perceived record. It is aimed to build such integrity constraints explicitly as an integral part of the canonical transform model.

The totality of the proposal can be better understood and appreciated by looking at how the transform model can be employed in practice. Descriptions of its data structures can be appropriately held in a data dictionary, with supporting software used in conjunction to define perceived record (types) and to retrieve corresponding view occurrences. The following diagram illustrates this. It must be emphasised that the three pieces of software represented below are at a logical level -- at implementational level, the various functions may be integrated as one package depending on the constraints.

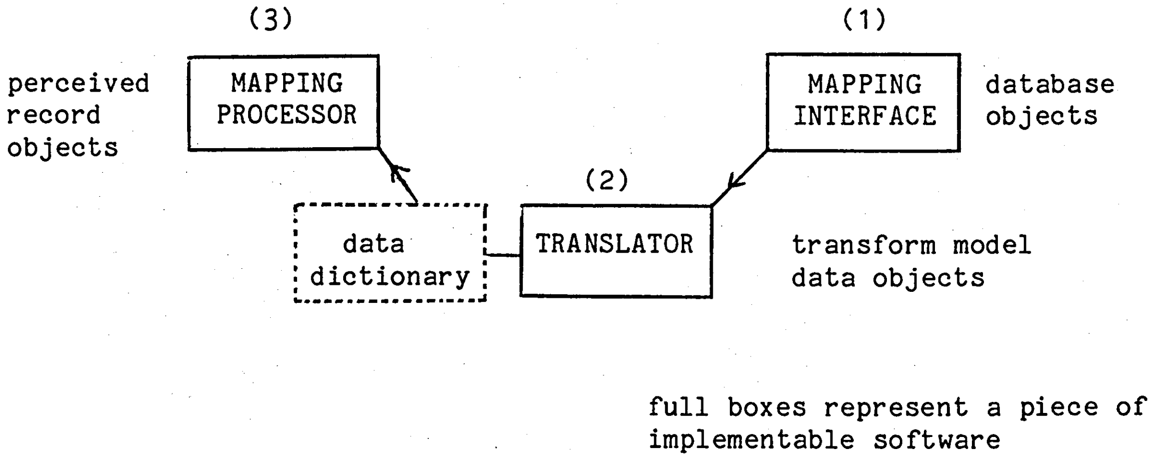


fig. 7.1 The Transform Model applied in a practical environment

The processes involved are :

1) The EUA defines a perceived record view in terms of database objects. The main function of the mapping interface is to validate the correctness of the specifications.
2) The translator translates the mapping definitions into equivalent descriptions of data structures in the transform model. These mapping descriptions plus those of the perceived record will be held in the data dictionary.
3) Upon an end user's valid request for a perceived record view, the mapping processor extracts the corresponding mapping descriptions (from the data dictionary) and uses them to retrieve the actual occurrences of the perceived record.

The interactions of the various types of data objects in the respective schemas are represented in fig. 7.2. Note that the logical schema is also included, only to show its association to the general EAR conceptual schema in practice.

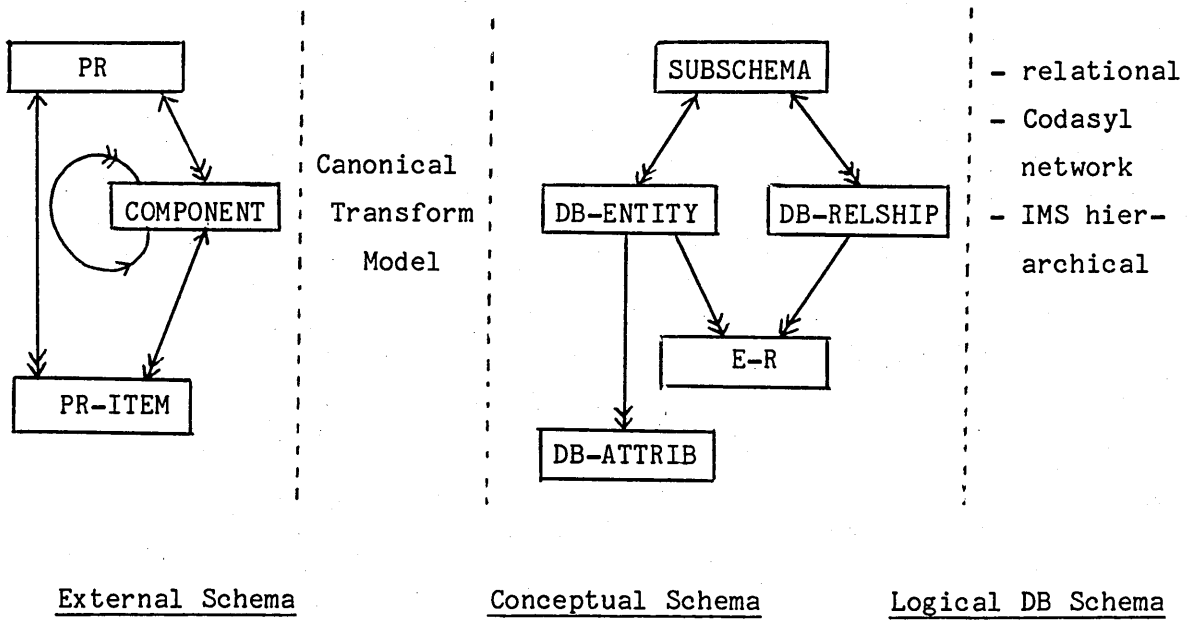


fig. 7.2 Schema Interfaces with the Canonical Transform Model

Subschema here defines a subset of the entire database which sufficiently contains the database objects that are required to construct a perceived record. The E-R (Entity-Relationship) data object represents the fact that one db-entity type can participate in many db-relationship types, and that, similarly one relationship may involve many db-entity types.

For the remaining parts of the chapter, the canonical transform model will be developed based on studying the requirements of database-to-perceived record mapping and applying the framework obtained in Chapter 3. The following section first formalises the underlying notions before embarking on the detailed modelling.

7.2 <u>The Formulation</u>

Hereinafter :

- o **component*** refers to the flat structure representing items of either a perceived record entity or a component entity in the perceived record model.

- o **X-object** (abbreviated to X-obj) refers to the canonical mapping object bearing the transformation characteristics of a component*.

The modelling is established at three levels. The first two concern the actual transformation of data objects, whilst the third is included so as to maintain the semantic integrity of the data occurrences in the mapping.

i) <u>At the data structure level pertaining to component* i.e. the perceived record or the component entity</u>
This involves two stages as illustrated below.

$$\begin{tikzcd} \text{Component}^{*} & \text{X-obj} \arrow[l, "(b)"'] & \text{Db-entity (linked by db-relationship)} \arrow[l, "(a)"'] \end{tikzcd}$$

Stage (a) describes the derivation of X-obj from db-entities. Stage (b) describes the construction of X-objs into the corresponding perceived record.

ii) <u>At the data value level, pertaining specifically to the pr-item</u>
For this, an elementary data mapping object called **Y-object** (Y-obj) is defined which describes the transformation characteristics of a pr-item.

The two stage structural correspondence between pr-item and db-attributes is as follows:

$$\begin{tikzcd} \text{Pr-item} & \text{Y-obj} \arrow[l, "(d)"'] & \text{Db-attribute or some other derivable property} \arrow[l, "(c)"'] \end{tikzcd}$$

Stage (c) describes the derivation of Y-obj from db-attributes. Stage (d) describes the correspondence of Y-objs to a pr-item.

iii) <u>At the data occurrence level</u>
The values of pr-items make up an instance or occurrence of a perceived record. As noted in Chapter 2, structural transformations cause a reorganisation of occurrences of db-entity i.e. the db-attribute values within. Value transformations applied to db-attributes must be defined on the correct instances of db-attribute values in order to ensure the integrity of perceived record occurrences. For this, the correct interaction of mapping information at the two higher levels of modelling are required.

In the next section, the design of the canonical transform model will feature the mapping descriptions required at each level. The model will be in 3NF.

7.3 <u>The Development</u>

Sections 7.3.1 through 7.3.3 consider the mapping at the levels of data structure, data value and data occurrence respectively. It is assumed that within a single database, db-entities and db-attributes can be uniquely identified by their respective names, represented by db-ename and db-attrname. Furthermore, a X-obj or Y-obj is termed basic if it relates directly to an existing data object described in the database schema. If the relation is expressed by some transformation rules, it is termed derived.

Capitalised names (e.g. X-OBJ) refer strictly to the schema data object in question; if otherwise, a general context is implied.

7.3.1 Correspondence of component* to db-entity

Accordingly, the study here involves the two stages identified in the earlier section.

7.3.1.1 Derivation of X-obj from db-entities

(I) When X-obj is basic -

The correspondence between X-obj and db-entity is many-to-one (m:1); each X-obj maps directly to one db-entity; each db-entity however relates to more than one X-obj.

(II) When X-obj is derived -

A derived X-obj can be mapped from more than one db-entity. In general, each db-entity may correspond to more than one X-obj that is either basic or derived. The simple m:1 relationship between X-obj and db-entity now generalises to a complex m:n. (See fig. 7.3)

The derivation is characterised by a structural transformation defined on the source db-entity. Chapter 3 gave five main classes of structure transforms -- concatenating, merging, subsetting, summarizing and transposing. Each of these can be specified by some kind of rules. The middle three transforms can be directly defined using an appropriate operator. Concatenating transforms, on the other hand, are slightly more complicated because they involve db-relationship types other than just db-entities in the process. The essence of transposing transforms lies primarily in the semantic-motivated mappings of pr-items; these, as such, will be examined in the next section.

We first look at the three operators and the modelling of the associated mapping information.

i) Mrgset

The Mrgset operation assumes the semantics of a merging transform. It takes two or more entity types and generates a single common class of entity.

ii) Subset

The Subset operation assumes the semantics of a subsetting transform. The selection criteria defined on attributes of the entity may consist of the comparison operators =, <>, >, >=, < and <= ; boolean operators AND, OR and NOT, and parentheses to indicate a desired order of evaluation.

iii) Grpset

The Grpset operation assumes the semantics of a summarizing operation. It must be specified with a grouping predicate defined on a given attribute of the source entity.

Information about each of the set operations associated with an X-obj derivation can be recorded using two attributes which describe the X-OBJ entity in the transform model. SOPTYPE (Structure Operation Type) defines the kind of operation; it has domain values of MRGSET, SUBSET and GRPSET. In order to generalise to basic X-obj, a fourth domain value, NIL is used to indicate the absence of transformations. A second attribute PRED (predicate) describes the qualification criteria that may be required for the process.

Whilst these two attributes can sufficiently model the operator-defined transformations, the modelling of concatenating transforms requires much deeper analysis. As seen in Chapter 3, a concatenating transform consists of two essential concepts : an origin node and a set of immediate arcs. In this case of X-obj derivation, the db-entity chosen for the role of an origin node will be referred to as the **anchor** (entity) of the X-OBJ. Each immediate arc is represented by a concatenating predicate expressed in terms of db-relationship connecting the db-entities.

It is necessary to be more specific about the meaning of the concatenating predicate here. This can be :

- o an explicit named db-relationship
- o a value-matching predicate that is defined on db-attributes belonging to the two 'nodal' db-entity types. This may express an existensial relationship such as in relational logical schemas -- the existence of an explicit relationship between two relations is defined by key propagation and the subsequent matching of the data item values. Alternatively, it may be used to derive an implicit relationship between two entities, as we have already seen with the LOOKUP operator in the Codasyl EUF DDL.

Earlier on, it was established that X-OBJ has a m:n relationship with DB-ENTITY. In a concatenating transform, the relationship between a given X-obj and each of its source db-entities must be fully described by a predicate which indicates the connection to be used to select the correct occurrence of the db-entity type, with respect to a predefined anchor entity. Note that when the source db-entity is in fact the anchor entity, this description does not apply. The complex m:n relationship is now resolved into two 1:n relationships by introducing an additional entity called XLINK as shown below.

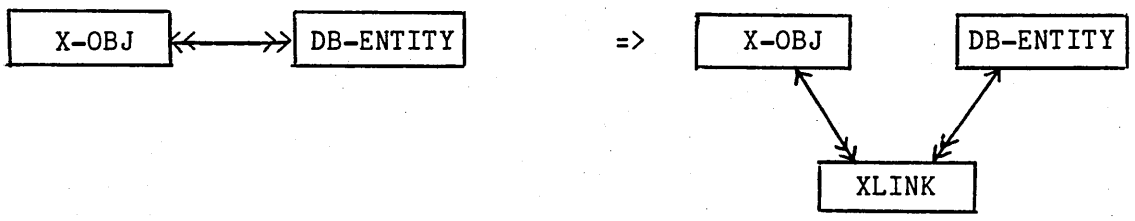


fig. 7.3 Relationship between X-OBJ and DB-ENTITY

XLINK has a composite key comprising of the two identifying attributes from X-OBJ and DB-ENTITY. It also has an additional attribute called LINFO (link information) which describes the concatenating predicate. LINFO will take one of the following forms:-

o r where r represents the name of a db-relationship in an immediate association between the db-entity in question and the anchor entity.

o multiples of a triple descriptor of type $\langle r_1 - e - r_2 \rangle$, separated by commas -- r_1 and r_2 are db-relationship names; e is a db-entity name. Such descriptors will be used to define complicated associations. For example, suppose an X-obj has sources of db-entity A and C as shown in the diagram. A is the anchor entity. LINFO description for an XLINK occurrence involving db-entity, C, will be expressed as :

$$\begin{tikzcd}
& A \arrow[d, "x"] & \\
& B \arrow[dl, "y"'] \arrow[dr, "z"] & \\
C & & F \arrow[ll, "t"]
\end{tikzcd}$$

(i) x - B - y or
(ii) x - B - z, z - F - t
depending on which predicate is required

fig. 7.4 Example of a Cyclic Schema

o $a_1 = a_2$ where a_1, a_2 are db-attributes of the db-entities involved.

o a null value - this happens in the case of when :
 i) the source db-entity is the anchor entity in a concatenating transform.
 -- in the above example, LINFO will be null for XLINK occurrence that is identified partially by the db-entity A.
 ii) the inter-structural transform is not a concatenation i.e. it is a merging transform.

For two reasons, the anchor entity name is explicitly associated with an X-obj by adding one more attribute to X-OBJ, appropriately referred to as ANCHOR. Its major purpose relates to the identification of X-obj occurrences in constructing a complex pr. A perhaps lesser reason is the ease of checking out the correctness of the LINFO description. With basic X-obj, or those described by a Subset or Grpset operator, the ANCHOR value defaults to the only source db-ename. Merged X-obj i.e.

that defined by a Mrgset operator, has optional ANCHOR values which can be either one of the multiple source db-enames. The ANCHOR value in this case will be indicated by the value 'OPT' which is system-defined to mean one of the associated source db-enames.

The attributes for the X-OBJ and XLINK entity in fig. 7.3, can now be gathered together. The X-OBJ attributes identified so far are SOPTYPE, PRED and ANCHOR. No one of these attributes, or a combination of them, can form an identifying key required for X-OBJ. It is therefore necessary to construct an internal identifier; this will be called X-ID (X-identifier). For the moment, it suffices to say that X-id is a value generated by an algorithm invoked each time a new X-obj is defined. The algorithm will be considered further as an implementational aspect. Since DB-ENTITY is identified by the attribute DB-ENAME, the composite key of XLINK consists of both X-ID and DB-ENAME.

The entities defined so far are now written down with their attributes. Unique identifiers are underlined.

X-OBJ (<u>X-id</u>, soptype, pred, anchor)
XLINK (<u>X-id</u>, <u>db-ename</u>, linfo)

The next stage now is the construction process of perceived records from X-objs.

7.3.1.2 Case of a simple perceived record

The correspondence here is fairly straightforward. A simple pr has no components and therefore relates to just one X-obj which may be basic or derived. Each X-obj on the other hand, may feature in one or more simple pr, and possibly other complex pr. A one-to-many (1:m) relationship hence exists between an X-OBJ and a PERCEIVED RECORD, as shown below.

$$\begin{tikzcd}\boxed{\text{PR}} \arrow[r, <<->] & \boxed{\text{X-OBJ}}\end{tikzcd}$$

fig. 7.4a Relationship between PERCEIVED RECORD and X-OBJ

In general, the pr-items in a simple pr pertain directly to the base entity i.e. they originate from the same db-entity type. This means that the end user could retrieve data directly from the database rather than via a perceived record. In other words, the provision of simple pr s (and maybe its concept) does not seem to be particularly meaningful, except for the aim to have a generalised facility. There are however two interesting situation when a simple pr maps from a derived X-obj where its pr-items are extracted from entity types related to the base entity type. It is probably these exceptions that justify the case for simple prs.

<u>Case (i) Inclusion of a summarized pr-item</u>

The required view information may consist of summarized data describing the user-perceived object, i.e. the base entity of the perceived record. Consider the following example featuring a portion of the HOSPITAL database :

$$\begin{tikzcd}\text{WARD(W-no, W-name)} \arrow[d, "\text{Registers}"] & \\ \text{PATIENT(P-no, P-name, Age, Sex)} & \end{tikzcd} \quad \Rightarrow \quad \text{WARD-VIEW(W-no, W-name, Tot-Pat)}$$

WARD-VIEW describes a ward and the total number of patients it has. The base entity here corresponds to the WARD entity in the database. The pr-item Tot-Pat is summarized from a count of occurrences of the PATIENT entity which is related to the base entity via the db-relationship Registers.

Case (ii) Composite keys

The perceived record base entity may correspond to a db-entity that can only be identified by db-attributes from its 'owner' db-entities. This happens when the db-entity contains intersection data resulting from the resolution of a complex m:n relationship between the two 'owner' db-entities. An example is as follows :

$$\begin{tikzcd}
\text{DOCTOR(D-name,Age, Sex)} \arrow[dr] & & \text{PATIENT(P-no,P-name, Sex ,Age)} \arrow[dl] \\
& \text{TREATMENT (Date, Drug-code)} &
\end{tikzcd}$$

=> TREATMENT-VIEW(P-no, D-name, P-name, Date, Drug-code)

Only a bare minimum of attributes have been used to describe the db-entities -- in practice, there will be more descriptive information held. D-name and P-no identifies the db-entity DOCTOR and PATIENT respectively. TREATMENT-VIEW in this case is required to have these two identifying attributes both for consistency reasons and to make it meaningful.

7.3.1.3 Construction of a complex pr

The process of constructing a complex pr is an assembly of X-objs that form the building block structures of the transform model. Recall that the term component[*] was adopted earlier in Section 6.2 to refer to either a perceived record entity or a component entity, that maps from an X-obj. This can now be expressed more precisely. Each component entity, in the same way as a perceived record entity, maps from one X-obj. Every X-obj corresponds to one or more component entity, as well as corresponding to one or more perceived record entity. X-obj then has a 1:n relationship with COMPONENT as depicted in the following diagram in fig. 7.5.

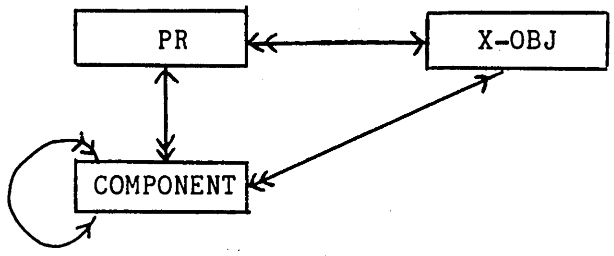


fig. 7.5 Relationship of X-OBJ to COMPONENT

More information however must be recorded about this assembling process. The semantics of a complex pr are expressed chiefly by the association of individual component to the simple pr i.e. base entity, and the associations between the various components. A crucial requirement is the selection of the correct occurrences of X-obj that match the expected semantics. This, in fact, is a second level of occurrence identification. The first, as seen in the previous section, concerns the identification of db-entity occurrences in a concatenating transform to derive an X-obj. The nature of the two are very similar, except that this next level is complicated by the fact that a complex pr may be a deeply nested structure.

The following first illustrates where the similarity lies, and expands on to take into account possible hierarchies of nestings. With reference to fig. 7.4,

- two complex pr types, with the same syntactic structure but different semantics, are constructed from the same basic A' and C'. A' and C' correspond to db-entity A, C respectively.

- PR-1 contains the meaning that component C is associated to base A by the associations x, y via db-entity B.

- PR-2 contains the meaning that component C is associated to base A by the associations x, z via db-entity B, and association t via db-entity F.

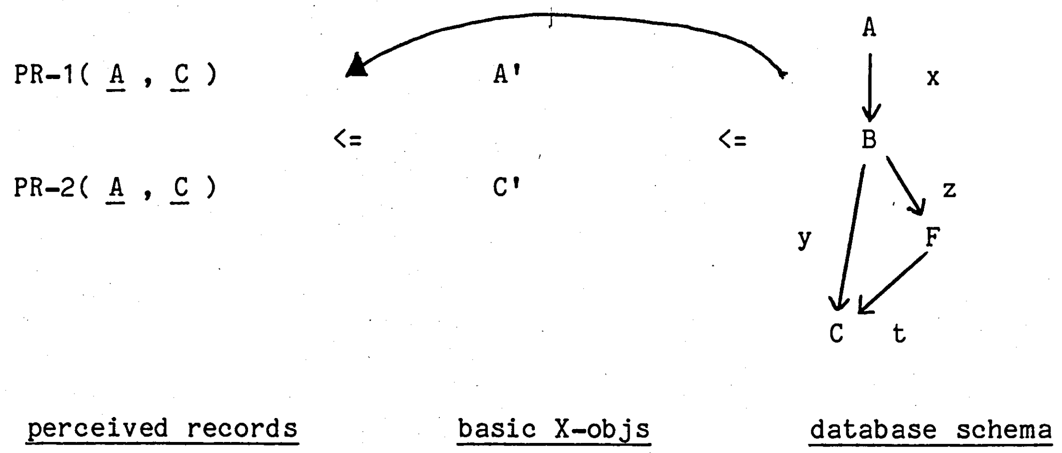


fig. 7.6 Identification of pr occurrences with different semantics

Notice that the X-obj A' and C' are defined only once. Note also that the direct process depicted by the single arrow $\leftarrow$, is no different from the example of deriving an X-obj with the same composition in Section 7.3.1.2. There, the predicates for determining the correct selection are expressed by means of the LINFO attribute in the XLINK entity.

The predicates, in this case, are described in terms of the anchor entities of the X-obj instead. The identification of a required X-obj occurrence of one type in relation to another X-obj type amounts to a step-wise procedure between pairs of the anchor entities of the X-objs involved. The information about which associations i.e. db-relationships will be used to relate the anchor pair pertain specifically to the given perceived record type, and forms part of its description. More precisely, the information is relevant to only the component entity which describes how the component ought to be derived with respect to the perceived record base entity. The perceived record entity, as such, does not contain explicit link information of this kind. For the purpose of modelling this aspect, the component entity has an extra attribute called C-DESC (Component-Description). C-DESC will be expressed in the same first three formats as defined for LINFO earlier. The fourth null value option of LINFO is not relevant here because the case of describing the linkage to the anchor entity itself is explicitly represented by the perceived record entity.

The following explains how the combined information ANCHOR and C-DESC information are used.

<u>Consideration of nested levels of components</u>

The above example illustrates a trivial identification of just one component occurrence to the base entity of the perceived record. In more complex situatuions, there must be a mechanism of keeping track of how pairs of nested components, more precisely their corresponding X-objs, can be identified correctly.

Each component in a nested structure belongs to a specific level (c.f. recursive tree structures in Chapter 4). All components at one level are identified to its parent component at one immediate level higher up. In the context of the perceived record model, the root of the nested hierarchy is represented by the perceived record entity.

We propose to express the embedded structural information explicitly by using an attribute LEVEL-NO which indicates the level position of a component in the hierarchy. The root conventionally has a level value 0 (zero). This however is implicit and the perceived reocrd entity does not carry the LEVEL-NO attribute. Immediate components have level value 1 and this increases with the depth of the structure. The component entity, hence has an additional attribute called LEVEL-NO, with domain values beginning from integer 1 to a desired limit (according to implementation considerations).

The example below, based on the same database schema as before, demonstrates the use of this piece of 'level' information in conjunction with the attributes ANCHOR and C-DESC, to fully define the X-obj occurrence identification process for a perceived record.
The perceived record here has base entity <u>A</u> and nested components <u>B</u>, <u>F</u> and <u>C</u>. Brackets indicate one or more occurrences of the data objects enclosed within each pair in the perceived reocrd representation.

PR($\underline{A}$ ($\underline{C}$), ($\underline{B}$ ($\underline{F}$))) <= A' B' C' F' <=

$$\begin{tikzcd}
 & A \arrow[d, "x"] & \\
 & B \arrow[ddl, "y"'] \arrow[dr, "z"] & \\
 & & F \arrow[dll, "t"] \\
C & &
\end{tikzcd}$$

perceived record — basic X-obj — database schema

fig. 7.7 Construction of a nested complex pr

The following steps are taken in constructing this perceived record :

i) The base entity corresponds to X-obj A'. The ANCHOR value of A' is thus extracted.

ii) The LEVEL-NO value indicates that $\underline{B}$ and $\underline{C}$ are first level components. ANCHOR values of corresponding X-obj B' and C' are then checked. C-DESC information from components for $\underline{B}$ and $\underline{C}$ are also extracted. Based on these, the anchors of components $\underline{B}$ and $\underline{C}$ are identified to their parent component anchor i.e. anchor of A' from step (i). (Both $\underline{B}$ and $\underline{C}$ will have null pc-name because of their immediate association to the base entity of the perceived record).

iii) Level-2 component is $\underline{F}$. Its parent component is $\underline{B}$. The ANCHOR values of F' and B' are then obtained. The correct selection of F' occurrence to B' occurrence is completed by knowing the anchor connection from the C-DESC information in $\underline{F}$.

The whole process can be specified by a generalised algorithm as follows :- For ease of notation, X-OBJ is parametrized to indicate its corespondence to a component* type. For instance, X-OBJ(COMPONENT) reads as X-OBJ corresponding to a COMPONENT.

```
Get ANCHOR of X-OBJ(PERCEIVED RECORD)
For LEVEL-NO =1 to specified-no-of-levels
  Begin
    For each COMPONENT at the level
      Begin
        Get ANCHOR of X-OBJ(COMPONENT) { call this a1 }
        Get C-DESC of COMPONENT
        If LEVEL-NO <> 1, then
        Begin
             Get PC-NAME of COMPONENT  { parent component }
             Get ANCHOR of X-OBJ( COMPONENT= parent component )
                                          { call this a2 }
        End
        else if LEVEL-NO=1  { do nothing }
        Connect a1, a2 by C-DESC of COMPONENT
      End  { for component }
   End { for level-no }
```

The attributes LEVEL-NO and C-DESC are now included in the appropriate entities. With reference to the sub-model of fig. 7.5, the additions are :

PR(<u>pr-name</u> , X-id)

COMP(<u>c-name</u>, <u>pr-name</u>, pc-name, level-no, c-desc, X-id)

The presence of X-id in both entities express their relationship with X-OBJ as shown in the diagram.

Fig. 7.8 sums up the entities of the transform model established so far, and their relationships with the other schema objects.

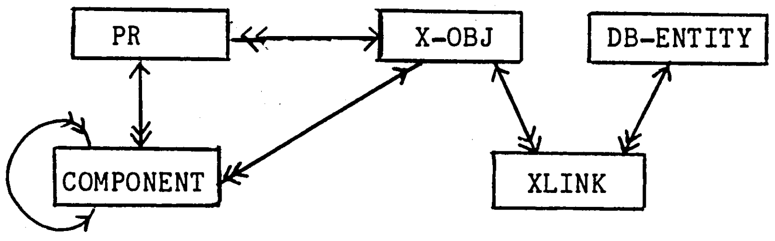


fig. 7.8 Mapping correspondences at Data Structure level

The next section now analyses the modelling requirements of data value transformation.

7.3.2 <u>Correspondence of pr-item to db-attribute</u>

The study is again divided into two parts.

7.3.2.1 <u>Derivation of Y-obj</u>

(I) When Y-obj is basic -

Each Y-obj relates directly to one db-attribute; there is no value transformation. Each db-attribute in turn may correspond to more than one Y-obj. The relationship between Y-OBJ and DB-ATTRIBUTE is thus m:1.

(II) When Y-obj is derived -

In terms of relating to db-attributes, a derived Y-obj maps from one or more db-attributes. A derived Y-obj may also correspond to properties such as that abstracted from the db-entity types or db-relationship types. For the moment, we focus on its association to db-attributes first. Each db-attribute, in general, relates to one or more Y-obj that may be basic or derived. As such, the general relationship between Y-OBJ and DB-ATTRIBUTE is in fact a complex m:n relationship. More information however needs to be recorded about this relationship.

A derived Y-obj defines a change in the source data type values as specified by a data item transformation rule. The various classes of item transformations have already been described in Chapter 3. We shall now define how information about these are modelled and recorded; in particular, specific operators will be defined where necessary.

Each transformation rule can be regarded as described by two attributes: IOPTYPE (Item Operation Type) which states the class of transformation, and SPEC (specification) which describes the complete specification of the operation. The domain values of IOPTYPE and the corresponding valid expressions for SPEC are as follows :-

IOPTYPE value of :

i) EXP (Expression) indicates the use of an arithmetic expression to evaluate a data value.
The corresponding SPEC expression consists of one of the first two, or a combination of all three, of the following fields, joined by arithmetic operators -- +, -, *, / with optional parentheses determining the order of evaluation :

- o db-attribute
- o summarized term on a db-attribute e.g. SUM(SALARY)
- o literal supplied externally as a parameter

ii) TAB(Table) indicates the 'look-up' of data values coded in a table
The corresponding SPEC expression has the form :

TAB-CODE = c-value

TAB-CODE identifies which table is to be accessed. C-value depends on how the storing of tables is implemented; it may be a numeric value.

iii) FCT(Function) indicates the use of a summarized function.
The corresponding SPEC expression is of the form :

FCT (db-attrname)   where FCT can be one of these:-
TOTAL, COUNT, MAX, MIN, AVE

iv) AGR(Aggregate) indicates an aggregation of more than one data value belonging to different db-attributes to a single data value.
The SPEC expression is :

AGR(db-attrname, ...)   where ... means item repeats

v) TRP(Transpose) symbolises a transposing transform accompanied by either one of the two item transformations defined in the SPEC expression.

- o MERGE(db-attrname , ...) produces a generic attribute. The operation specifies a linear to repeating transposition. (Refer back to Section 3.2.2.5).

- o a repeating to linear transposition is obtained by the SPEC expression,

$$\text{SPLIT(db-attrname WHERE } \left\{ \begin{array}{l} \text{ORDER = ' integer '} \\ \text{CHARATTR = ' value IN db-attrname '} \end{array} \right\} \text{)}$$

where { } indicates choice of item

The SPLIT operation generates a new attribute type with a data value that is defined in the qualification. The qualification ORDER specifies the position of data value occurring in a repeating structure. The convention adopted here is an ascending integer value. Generally then, given an attribute(type) A with values a_1, a_2, ... a_i in a repeating structure, appearing as :

A	
a_1	the order value 1 will pick out a_1,
a_2	the order value 2 will pick out a_2, and so on.
.	
.	
a_i	

CHARATTR denotes the attribute type that characterises the main attribute. (See Section 3.2.2.5 Case II).

Note that the semantic naming of the resultant attribute (which will form the pr-item) depends totally on the user who specifies the mapping; in this case, it is the EUA. For instance, in the example illustrated on page 43, the view data item name Eye-R will in fact be defined by the EUA using the following specification :

SPLIT(R WHERE CHARATTR = 'eye' IN 'Type')

Accordingly, the way that this operation has been specified requires prior knowledge of the underlying structure and data values. It is possible to define the operation in a declarative manner such that the number of attributes and associated attribute names can be generated without user-supplied values. This may be in the form :

$$\text{SPLIT(db-attrname } \left\{ \begin{array}{l} \text{BY-ORDER} \\ \text{BY-CHARATTR-name} \end{array} \right\} \text{)}$$

CHARATTR-name here is provided by the EUA. The resultant attributes implicitly will have names formed from combining the main attribute name with the order value or charattr value.

Despite the advantages of non-procedurality and no pre-knowledge requirement, the first specification is still preferred because :--

- o the mapping is defined by the EUA who is familiar with the underlying database; ease of non-procedurality is not significant.
- o it is easier to check out errors. In the latter case, the EUA has no idea nor control whatsoever of the kinds of attributes that may be obtained.

vi) PROJ (Project) symbolises a semantic-motivated mapping from a db-entity type or a db-relationship type.

The corresponding SPEC expression may be one of the following types of operations :

- o E-PROJ(db-entity-name, [db-entity-name, ...])
- o R-PROJ(db-relationship-name, [db-relationship-name, ...])

where [ ] indicates optional item

The first specifies mapping of an entity type to a data item value. More than one db-entity-names may be used when the mapping occurs in conjunction with a merging transform, and it defines a new attribute in the merged entity (such as the example of JOB-TITLE on page 31). The second operator specifies the mapping of a db-relationship type to a data item value.

vii)NIL is used in order to generalise the modelling to basic X-obj. The corresponding SPEC expression is a null expression.

The two attributes of IOPTYPE and SPEC describing the transformation rules associated in the derivation of Y-obj will be part of the Y-OBJ entity. The complex m:n relationship between Y-OBJ and DB-ATTRIBUTE will be further decomposed to produce normalised entities. Just as before, a 'link' entity is introduced. Y-OBJ has a 1:n relationship with YLINK; similarly the relationship between DB-ATTRIBUTE and YLINK is 1:n, as fig. 7.9 shows. YLINK consists of only two attributes which are the identifying attributes from Y-OBJ and DB-ATTRIBUTE, and which form its composite key. The attributes of Y-OBJ are just IOPTYPE and SPEC. These composite key. The attributes of Y-OBJ are just IOPTYPE and SPEC. These two either individually on their own, or combined together cannot be a unique identifying key for Y-OBJ. As with X-OBJ, an internal identifier, Y-ID(Y-identifier) is constructed for the purpose. The algorithm for obtaining Y-id value will be considered in the next chapter.

7.3.2.2 Pr-item mapping from Y-id

Each pr-item maps directly from only one Y-obj; each Y-obj however may correspond to more than one pr-item pertaining to different perceived record types. The relationship of Y-OBJ to PR-ITEM is thus m:1 relationship, as depicted below.

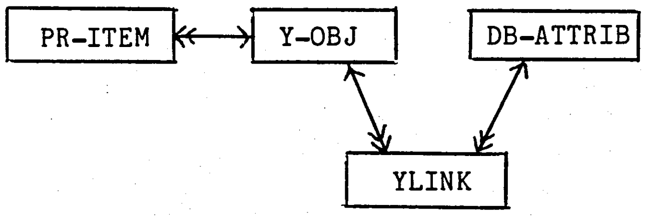


fig. 7.9 Mapping correspondences at Data Value level

The corresponding attributes and entities are as follows:

Y-OBJ(<u>Y-id</u>, ioptype, spec)

YLINK(<u>Y-id</u>, <u>db-attrname</u>)

PR-ITEM(<u>pr-itemname</u>, <u>pr-name</u>, c-name, pri-role, pri-format, Y-id)

Note the addition of Y-id to PR-ITEM in order to represent the m:1 relationship between Y-OBJ and PR-ITEM.

The sub-model obtained at the end of this section of modelling is now merged with that in the previous section (fig. 7.8) to present a more complete picture of the inter-connections between the data objects modelled so far. This is shown below.

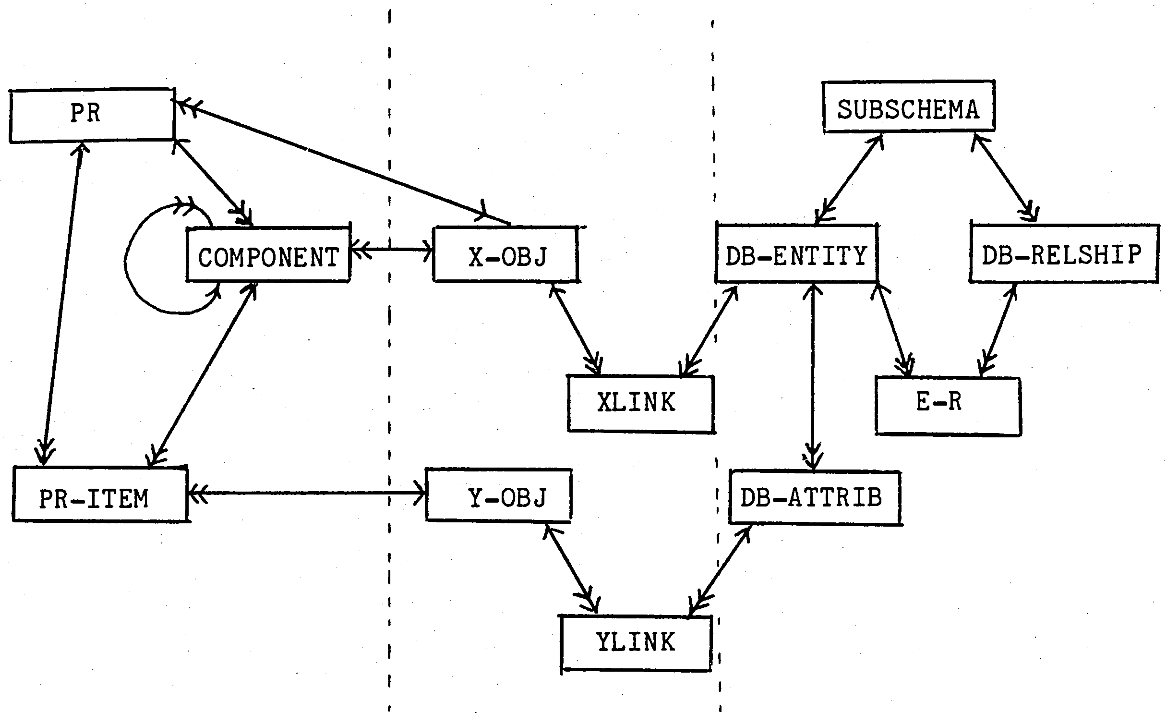


fig. 7.10 Modelling Data Structure and Data Item Transformation

The next section now extends to include semantic constraints that are required at the data occurrence level when the two levels of transformation interact.

7.3.3 Data occurrence correspondence

Each perceived record entity occurrence or each component entity occurrence maps from one corresponding X-obj occurrence which in turn is equivalent to one of the following :-

- o an original i.e. unchanged, db-entity occurrence (case X-obj basic).
- o a restructured occurrence consisting of one unique anchor entity occurrence, and possibly associated occurrences from different db-entity types that may repeat with same or different values (case X-obj derived).

The restructured occurrences may be one of these kinds :-

i) concatenated set of db-entity occurrences

ii) merged set of db-entity occurrences

iii) subsetted set of db-entity occurrences
iv) summarized set of db-entity occurrences
v) transposed set of db-entity occurrences

Notice how in Section 7.3.1 concatenating transforms have been modelled separately from other operator-defined transforms. It is possible to superimpose operator-defined intra-structural transforms on top of concatenating transforms, in one single piece of mapping specification. Hence, in addition to the above five categories, three more variants exist as follows :-

vi) concatenated + subsetted set of db-entity occurrences
vii) concatenated + summarized set of db-entity occurrences
viii) concatenated + transposed set of db-entity occurrences

Superimposition of intra-structural transforms on a merged set of db-entity occurrences cannot be obtained in a similarly straight manner. This would require the specification of more powerful operators that can combine the semantics of an intra-structural transformation and merging transform. As it is, with this present way of modelling, the task must be achieved in two steps : first define a merging transform, and then use the resultant object to specify a second required operation. The equation of the data occurrences are shown graphically below.

Equation (I)

$$\text{COMPONENT}^{*}\text{ occurrence} = \text{basic X-OBJ occurrence} = \text{DB-ENTITY occurrence}$$

Equation (II)

$$\text{COMPONENT}^{*}\text{occurrence} = \text{derived X-OBJ occurrence} \overset{\text{transformed}}{\subseteq} \text{DB-ENTITY occurrence}$$

Transformation rules defined directly on the db-attributes modify the values in a db-entity occurrence to form the pr-item values making up a perceived record occurrence. This definition satisfies Equation(I) above since the middle term of X-obj may be omitted altogether. It however fails with Equation(II); here value modifications must necessarily operate on the derived X-obj occurrence, which instead of being the original db-entity occurrence, is a restructured db-entity occurrence.

Summarizing transforms provide an illustrative case where functions are used in combination to summarize data values in sets of reorganised db-entity occurrences. In other cases, the data values may originate from one db-entity occurrence and its associated occurrences of related db-entity types as derived with concatenating transforms.

The direct relationship of Y-OBJ to DB-ATTRIB in fig. 7.10 subsequently is consistent in situations governed by Equation(I) but not Equation(II). This inadequacy is overcome by modelling the data occurrences of X-obj explicitly such that Y-OBJ occurrence values relate to those in X-OBJ occurrences. For this, we introduce a new entity X-ELEM (X-element). X-elements are elementary data attributes that belong to an X-obj; the set of X-ELEM values collectively form an X-obj occurrence. Each X-element type has, and is equivalent to a corresponding db-attribute type; this has an associated name, X-ELEM-NAME, which is appropriately a db-attrname. Y-OBJ, instead of relating directly to DB-ATTRIB, now maps from X-ELEM.

The sub-model of fig. 7.9 is reconfigured as follows :

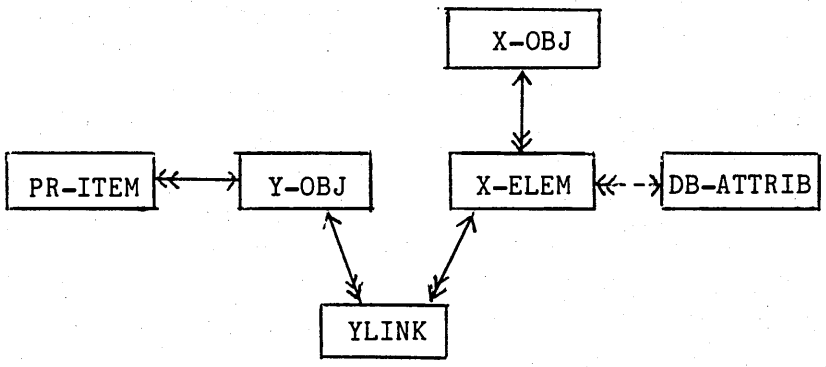


fig. 7.11 Modelling Integrity Constraint

Note the implicit m:1 relationship drawn in dotted lines between X-ELEM and DB-ATTRIB indicating the replication of db-attrname in X-ELEM-NAME.

Finally, the diagram of fig. 7.10 illustrating the first two levels of mapping information, can now be expanded to include this third level of mapping constraints. This is shown in fig. 7.12.

The attributes of the various data objects of both the perceived record model(these include those with mapping characteristics) and the transform model in fig. 7.9 are :

PR(<u>pr-name</u> , X-id)
COMP(<u>c-name</u> , <u>pr-name</u> , pc-name, level-no, c-desc, X-id)
PR-ITEM(<u>pr-itemname</u>, <u>pr-name</u>, c-name, pri-role, pri-format, Y-id)

X-OBJ(<u>X-id</u>, soptype, pred, anchor)
X-LINK(<u>X-id</u>, <u>db-ename</u>, linfo)
Y-OBJ(<u>Y-id</u>, ioptype, opexp)
X-ELEM(<u>X-elem-name</u>, X-id)
Y-LINK(<u>Y-id</u>, <u>X-elem-name</u>)

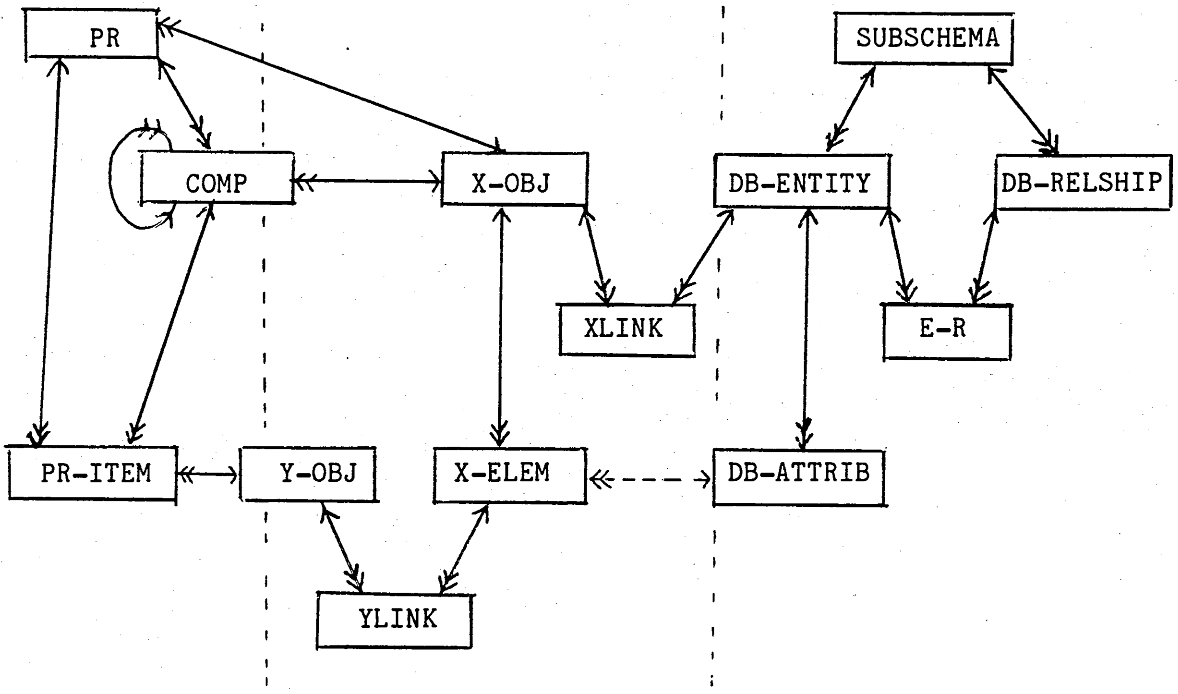


fig. 7.12 The Canonical Transform Model and its relationship to the perceived record model and database schema

7.4 Appraisal of the model

Advantages

The transform model has two benefits in terms of supporting derivations of many perceived records of different types from a database. The use of one basic type of data structure i.e. X-obj that is common to the mapping process leads to :

(i) A reduction in the complexities and efforts involved --

An X-obj(type) can be defined once and used to map various perceived records which share common characteristics without having to specify the mapping all over again.

(ii)An extended range of complex perceived records that can be mapped --

Many different complex views can be constructed by combining X-obj together in multiple ways (within the semantic constraints of the database). This advantage of flexibility is analogous to that of the binary relation model as demonstrated by Bracchi et al [74], Bracchi et al [76] and Pelagatti et al [7?] in the context of the entire system architecture.

Disadvantages

In terms of criticisms, two general points can be made :

(i) consistency requirements -

A significant amount of consistency checks must be built into the model. Validations must be carried out at the interfaces of all three schemas, between :

- o database schema objects and mapping data objects - these involve firstly, the consistency of X-obj with DB-ENTITY established via XLINK, and secondly the consistency of X-ELEM with DB-ATTRIBUTE.
- o mapping data objects and external schema schema objects - the required consistencies are between PERCEIVED RECORD/COMPONENT and X-OBJ, as well as between PR-ITEM and Y-OBJ.

The checks at this level comprise of general operations such as :

- making sure that the named database objects are valid ones.
- making sure that the attributes defined in an item transformation rule are compatible. The dbattribute COLOUR, for instance cannot be used in

an arithmetic computation.

At an integral level, there is also the requirement of semantic consistency of all the constituent components making up an entire perceived record. The validations here must take into account the important element of semantic equivalence.

(ii) excessive efforts required for simple, direct mappings -

The use of X-OBJ can be regarded as irrelevant in deriving simple perceived records which can be mapped directly from the database objects. Similarly, the explicit representations of X-ELEM in such cases, is not necessary. These extraneous procedures however, are inevitable if a generalised mechanism is the goal.

Chapter 8 IMPLEMENTATION OF A MAPPING INTERFACE FOR PERCEIVED RECORD DEFINITION

This chapter concerns the presentation of a mapping interface -- the Perceived Record Interactive Mapping Controller (PRIMC) -- in the form of an interactive system that controls the specification of perceived records by the EUA.

The intent for building such a facility is three-fold :

- o to evaluate the kinds of consistency checks required, and hence the practical feasibility of the proposed transform model mapping mechanism.
- o to minimise the possibilities of deriving views with semantics that do not match expected semantics of end users.
- o to ease the task of specification and hence reducing the probability of errors. It must be stressed here though, that the design of a 'very' user-friendly interface (such as will be desired for casual users) is not an immediate concern.

The first two objectives, in particular, require a very careful design of the interface and its capabilities. Section 8.1 gives an overall view of the interface and its interconnection to other components in a query system environment. Section 8.2 addresses the principal feature which is the mapping definition language. The ultimate aim is the feasibility of the proposed language in terms of translating the language declarations into structures of the transform model. Section 8.3 specifically examines this aspect. An implementation based on a relational database has been developed. Section 8.4 reports on the current state of the implementation.

8.1 An overview

8.1.1 Nature of the interface

The interface is designed to aid specifically the EUA to define perceived record views of a database. It provides an automated set-up of the structure declarations of the mapping definition language and prompts the EUA for actual data value input at the appropriate places. The entire system is driven by a piece of control software; when inconsistencies are detected, it issues error messages; where necessary, it prescribes corrective measures to the EUA. To present a clearer understanding of its use, where possible, generic terms that occur in later parts of the specification will be replaced by the corresponding data names input (by the EUA) earlier during the session.

At the end of each session of perceived record definition, data files consisting of the perceived record object descriptions and transform model descriptions will be generated. In other words, the database mapping specifications are automatically translated into the corresponding transform model descriptions.

For any user to be able to specify mapping requirements (the EUA or DBA inclusive), he or she must learn and understand the syntax of the mapping definition language. The provision of a system that sets up the actual syntactic structures of the mapping definition language helps to alleviate the user, specifically the EUA to a large extent, from the complexities and syntactic concerns of coding a view definition. It not only simplifies the task but has the advantage of eliminating the need to check for syntactic errors in writing down the clauses of the language that are often made by the human user.

A similar kind of facility was reported by Bell [81] which concerns the use of Codasyl-typed DBMS. Called IDBACS (Interactive DataBase Administrator Control System), it relieves the DBA from having to specify the various DDLs (Data Definition Languages) correctly. The DBA faces interactive sessions of questions and answers during which data files are created for the construction of an I-D-S/II database. We however believe that the generation of the complete language clauses and statements is more appropriate than a simple style of questions and answers. This, firstly, is in view of the fact that the EUA is not in the least an ordinary user; secondly, such an approach still maintains the overall flavour of the language.

8.1.2 Overall architecture

Fig. 8.1 shows the relationship of the PRIMC facility with other components of a database system.

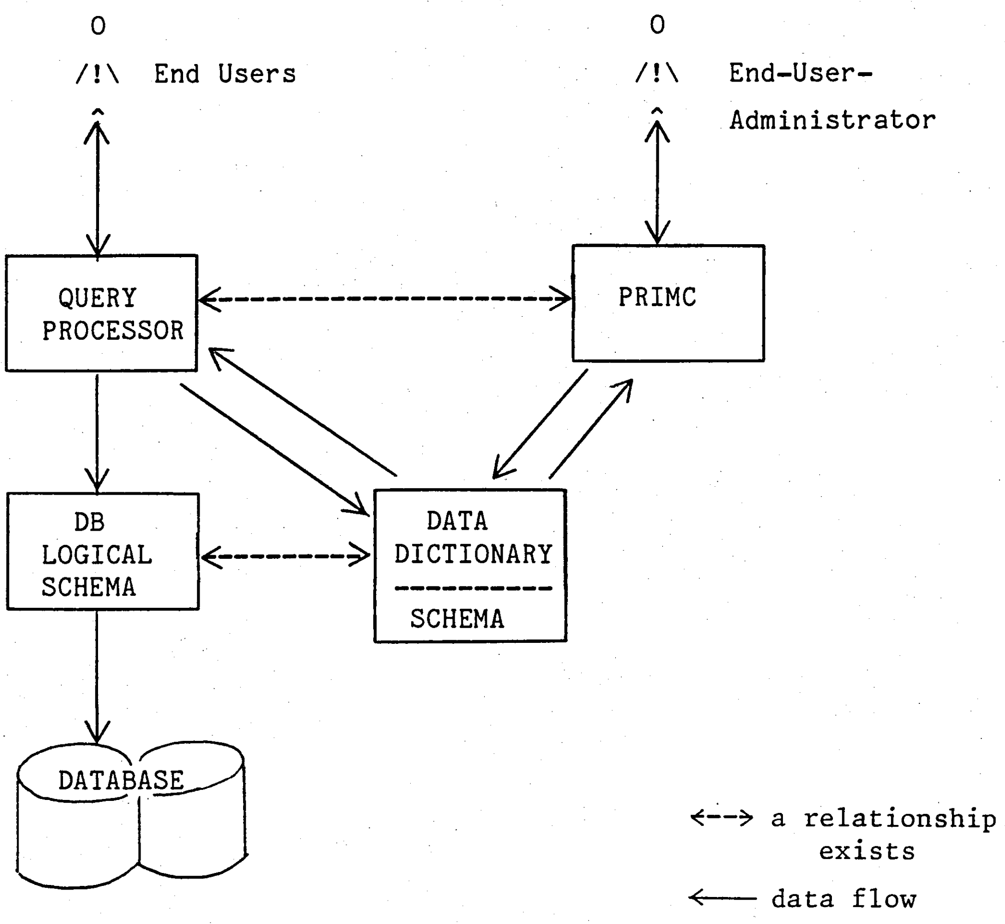


fig. 8.1 Overall query architecture

Note that PRIMC is separated from the query processor. Recall the three boxes of software -- the mapping interface, translator and mapping processor, in fig. 7.1. PRIMC effectively encodes the first two functions, whilst the third is managed by the query processor.

The query processor's main function is to compile and interprete end user queries and to retrieve the actual data occurrences in response; the occurrences, in this case are perceived record occurrences. In general, the query procesor may also optimise the actual access paths used to retrieve the data occurrences. This decision rests mainly with the implementation and design of the entire query system, and thus is not our concern here. For ease and independence of implementation, the component of PRIMC forms a stand-alone piece here. It however can be integrated into the query processor box given that the latter is capable of manipulating and retrieving perceived record occurrences. The dotted arrow between these two boxes in the diagram illustrates this.

The data dictionary component plays an important role here. In practice, descriptions of the database schema are also held in the data dictionary. It acts as the chief communication block between PRIMC and the database. Validations of all database object descriptions are made with the data dictionary. Furthermore, it is the repository of all the perceived record definitions and the transform model object descriptions. This however is a theoretical basis. Quite often in actual implementations, this may not work out nicely, either because no appropriate data dictionary facilities are available (as happens in this particular research) or due to other kinds of limitations.

8.2 A language for perceived record definition

The chief purpose of the mapping definition language is to define a perceived record and its semantics, and to specify the inter-relationships between it and the database structures.

The language must feature (at least) three distinct levels of details :-

1) Syntactic naming of the perceived record data types.
2) The expression and correct translation of end-user requirements of the view semantics in terms of the associations between the various components which make up a perceived record.
3) Database related mapping definitions which involve the declarations of the structural and item transformation rules.

Several factors have influenced the way the language is designed. Section 8.2.1 considers these in detail. Section 8.2.2 gives the meta language for the subsequent presentation of the entire language and its syntax in Section 8.2.3.

8.2.1 Design considerations

One of the foremost issues concerns the clarity and conciseness of the language. Structuring of the language into parts or sections, referred to in this thesis as blocks, each dealing with a specific functionality helps to improve the understandability and its usage. Accordingly, the approach adopted is a modular one; for each of the three levels of requirements named above, there is one corresponding main block of declaration in the body of the language.

Non-procedurality forms another major consideration. However, this factor must be weighed out with the important need to optimise validations in an interactive two-way, 'system-and-user' feedback environment.

The procedurality of a language in general, is largely characterised by the amount of sequencing [Leavenworth and Sammet 74]. Sequencing here refers to the order in which the steps necessary to carry out a task must obey. The general description of non-procedurality as stating what is wanted instead of how to go about achieving it, can be looked at from the point of view of independence from any arbitrary sequencing requirements. Two notions of sequencing can be identified : sequencing within one statement, and sequencing across a number of statements. The order of evaluation in an arithmetic expression, as dictated by the precedence, if any among its operators, provides a good example of the former. The latter can be generalised to blocks comprising of multiple statements. It is to this second form of sequencing that we shall pay particular attention.

An interactive facility controlling the specification in a step-wise manner requires that inconsistencies be spotted and put right as early as possible. Any late detection that demands a repeat (and possibly many times more) of previous sections of the specification in order to correct the errors is totally unacceptable. It would be most undesirable in terms of user-convenience and certainly would produce disastrous effects in terms of operational efficiency. In addition, such a possibility could strain the already large volume of built-in consistency requirements of the transform model. This problem therefore commands a very careful and thorough consideration of the various dependencies inherent in the definition of a perceived record and its mapping from the database.

Such dependencies, expressed in terms of the language constructs, can occur at two levels : those which inter-relate between blocks of statements, and those which exist amongst the statements within each block.

In general, each block couples to one another in some ways; this can be thought of as a kind of logical input-output connection. The output information from the precedent block becomes the input of the next, forming a cascade as depicted in fig. 8.1 below. The flow of information consists of data needed for the validation of externally-provided values entered by the EUA.

$$\begin{tikzcd}
\text{BLOCK 1} \arrow[r, circ-to] & \text{BLOCK 2} \arrow[r, circ-to] & \text{BLOCK 3} \arrow[r, circ-to] & {}
\end{tikzcd}$$

$\circ\!\!\longrightarrow$ input , output of information

fig. 8.2 Dependencies between Blocks of Statements

The sequence of the three blocks as specified in the syntax of the language results from the the inter-coupling properties that require data to be obtained in a specific order. Furthermore, it is only with a correct sequencing that the generation of known data names during the course of specification in place of generic terms, is possible.

It follows that much though the language is desired to be as declarative as possible, an element of procedurality necessarily prevails. On the other hand, the constraint of sequencing does not significantly affect the EUA since the specification is actually controlled and guided by the system.

8.2.2 The meta language

The formalisms used for presenting the complete syntax of the mapping definition language are as follows :

1) Capitalised letters represent keywords e.g. IS-FROM.
2) Lower-case letters represent generic terms, e.g. prname, which will be replaced by specific user-supplied names.
3) Curly brackets indicate choice of several options e.g. $\left\{\begin{array}{l}\text{INTEGER}\\ \text{REAL}\\ \text{CHAR}\end{array}\right\}$
4) Round brackets indicate optional items that may be omitted e.g. (COMP cname, ...)
5) Ellipses indicate possible repetition e.g. dbename, ...
6) Square brackets represent sub-blocks consisting of several statements e.g. $\left[\begin{array}{l}\text{IS-FROM}\\ \text{FORMAT IS}\end{array}\right]$
7) A slash indicates either one of <u>two</u> choices e.g. PR IS prname/ COMP IS cname

8.2.3 Structure of the language

Following from Section 8.2.1, the proposed language consists of three main blocks, in the sequence of DECLARATION, MAPPING and INTER-COMP-DESCRIPTION. Each block will be explained in turn.

8.2.3.1 DECLARATION

This block specifies the name of the perceived record, the names of its pr-items and components and the whole of the perceived record structure. A component construct is declared in the same way as a simple pr with no components; it has the following generic form contained within the block heading :

DECLARATION.

```
        PR IS prname/ COMP IS cname (REPEATS)
           HAS   ┌PR-ITEMS  pritemname, ...┐
                 │( COMP    cname , ...   )│
                 └                         ┘
```

<u>Some notes</u> :

- o COMP simply abbreviates COMPONENT.
- o The term REPEATS indicates whether a component is a repeating group. Its omission implys a non-repeating group.

<u>Rules</u> :

- o Pritemname(s) and cname(s) occurring in a HAS block belong immediately to the preceding declared structure of either PR or COMP. This means they are not further contained in a sub-structure. For example, any declared cname belonging to a PR declaration pertains to the first level or immediate components of the perceived record.

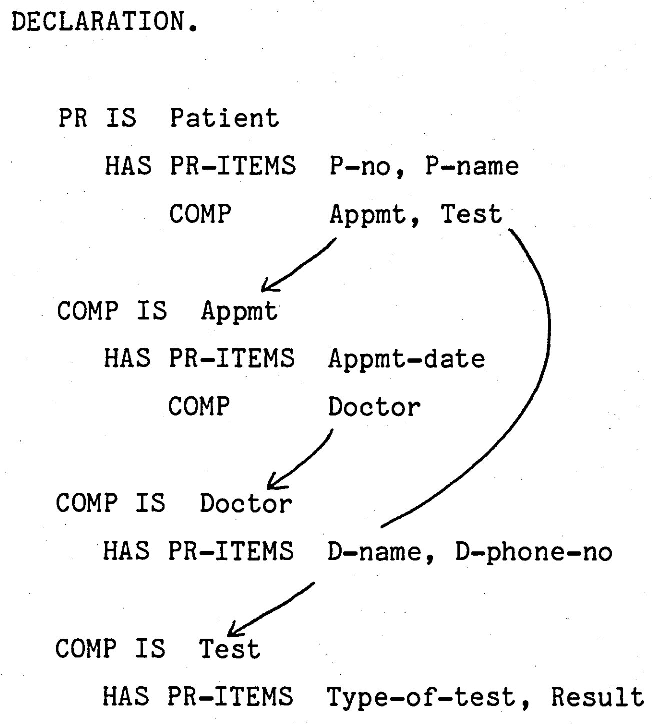
- o Each cname specified in the HAS block must be fully defined by a corresponding COMP DECLARATION block in the order that they appear in the 'HAS' list. If a component contains sub-components, each level of component in the nested structure will be fully declared until no more nestings exist, before specifying the contents of the next cname in the original list. The following example will illustrate what this means.

  Suppose a perceived record has two first level components A and B with the nestings represented linearly as follows :

  A (C (E (F))), B (D)

  F and D are simple sub-components with no children components of their own. The 'HAS' list of the PR DECLARATION has cnames A, B. The subsequent order in which the components are specified will be A, C, E and F. Upon completion of the first embedded structure, B and D will then be defined.

A practical example should further clarify this :
For the PATIENT perceived record view used throughout in this thesis, the DECLARATION specifications corresponding to its linear representation given on page 8 of Chapter 1, is as follows :



```
DECLARATION.

    PR IS  Patient
       HAS PR-ITEMS  P-no, P-name
           COMP      Appmt, Test

    COMP IS  Appmt
       HAS PR-ITEMS  Appmt-date
           COMP      Doctor

    COMP IS  Doctor
       HAS PR-ITEMS  D-name, D-phone-no

    COMP IS  Test
       HAS PR-ITEMS  Type-of-test, Result
```

The arrows are drawn to indicate the sequence of specification linking the provided cnames and the associated declarations. In the context of the interface, the EUA would need to specify <u>once</u> the order in which the components are embedded in the DECLARATION section. In the rest of the specification, the control system will generate the appropriate statements according to the defined order.

8.2.3.2 <u>MAPPING</u>

The MAPPING block defines the derivation of perceived record objects specified in the DECLARATION block, in terms of database objects. The definition includes that of :

- o a perceived record entity or component entity i.e. a specification of the corresponding X-obj.
- o pr-items belonging to the perceived entity or component entity i.e. a

specification of the corresponding Y-objs.

Accordingly, the overall syntax is partitioned into two parts for each of the purposes.

MAPPING.

```
  PR prname / COMP cname
   ┌                                                  ┐
   │ IS-FROM BASE    dbename                          │
   │                                                  │
   │ ({OR-FROM BASE    dbename                      }) │      Part I
   │ ({[AND-FROM BASE  dbename   WHERE *  ], ...    }) │
   │                                                  │
   │ HAS OPERATION *{      }                          │
   │                                                  │
   │ PR-ITEM  pritemname *[      ]                    │      Part II
   │                                                  │
   └                                                  ┘

                                                   *  to be expanded
                                                      later
```

<u>Some general points :</u>

- o The order in which the COMP MAPPING definition is specified follows from that given in the DECLARATION block.
- o In the interactive controlled specification, the generic names of prname, cname and pritemname will be automatically generated by the system using the information entered in the first block of declarations.
- o The deliberate containment of pr-item mapping specification within a PR/COMP specification block establishes a very useful consistency constraint -- all db-attributes named must originate from the named db-entities.

The rest of this section now examines in detail the statements in Parts I and II.

Part I

The IS-FROM / OR-FROM / AND-FROM statements specify the source db-enames. The option OR-FROM denotes the generalisation of a common class of view entity; no explicit database relationship is required to qualify this choice. This clause could have been combined in the IS-FROM clause in which case there would be more than one db-enames. The reason for precisely separating it out is for sake of clarity and conciseness. Note that the corresponding operation must be MRGSET.

The option AND-FROM embodies a concatenating transform. The clause may be repeated to accomodate the specification of more than one db-entity to be related to the base entity. The WHERE clause has the following form :

```
WHERE LINK-IS ⎧ dbrelationship                                    ⎫
              ⎩ INTERM  db-ename  WHERE LINKS-IS  db-relship, ... ⎭
```

The clause defines the concatenating predicates to be used. The first choice of db-relship indicates a simple, direct association between the base entity and the other relevant entity. An indirect linkage is specified by the second option; the term INTERM abbreviates INTERMEDIATE. In this case, (at least) two db-relship descriptors must be provided -- the order in which they appear is not relevant. We felt it reasonable to assume that in most circumstances, one intermediate is usually used. (For an example of intermediate linkages, refer back to fig. 7.4 on page 110 of the previous chapter). It must be stressed though, that a named intermediate db-entity serves only as an essential bridging object; its contents of db-attributes cannot be extracted for use in a pr-item transformation.

The reserved word BASE has been used to refer to both the base entity for a given perceived record entity type, and its synonymously-associated concept in the context of a component entity type. (This is in view of the fact that both are structurally-equivalent and is expressed by an X-obj).

The HAS OPERATION sub-block defines the type of structure transform operation required. Based on the list described in Section 7.3.1.1, the HAS OPERATION sub-block has a format as follows :

```
HAS OPERATION ⎧ MRGSET                                 ⎫
              ⎨ SUBSET  db-attrname  # data value      ⎬
              ⎪ GRPSET  db-attrname (# data value )    ⎪
              ⎩ NIL                                    ⎭
```

# stands for any of the comparison operators =, >, >=, <, <= . The data value for the equal comparator (=) may be a character string; the rest of the options are only valid with numeric data values. Note the bracketted term after db-attrname in GRPSET. The db-attrname here defines the grouping predicate and this may be optionally qualified by some criteria if desired.

Part II

The PR-ITEM sub-block specifies :

o the source db-attributes and any required transformation rule.
o whether the pr-item forms a prkey for the perceived record.
o the format of the pr-item.

Its syntax looks like this :

```
PR-ITEM  pritemname  ⎡ IS-FROM ⎧ db-attrname ⎫  ⎤
                     ⎢         ⎪ EXP         ⎪  ⎥
                     ⎢         ⎪ TAB         ⎪  ⎥
                     ⎢         ⎨ FCT         ⎬  ⎥
                     ⎢         ⎪ AGR         ⎪  ⎥
                     ⎢         ⎪ MERGE       ⎪  ⎥
                     ⎢         ⎪ SPLIT       ⎪  ⎥
                     ⎢         ⎪ E-PROJ      ⎪  ⎥
                     ⎢         ⎩ R-PROJ      ⎭  ⎥
                     ⎢                          ⎥
                     ⎢  (IS-PRKEY)              ⎥
                     ⎢  FORMAT ⎧ INTEGER ⎫      ⎥
                     ⎢         ⎨ REAL    ⎬      ⎥
                     ⎢         ⎩ CHAR    ⎭      ⎥
                     ⎣                          ⎦
```

The IS-FROM sub-block has a list of options in accordance with the descriptions of the operators given in Section 7.3.2.1. The first choice of db-attrname signifies a direct one-to-one mapping from the database.

The IS-PRKEY clause, when omitted indicates that the pr-item is not an identifying attribute for the perceived record. The FORMAT clause is self-explanatory. It does appear that the placement of these two clauses in the MAPPING block seems most inappropriate; they ought to feature in the DECLARATION block since they are properties pertaining to the perceived record description. The reason here being that with prior knowledge of its source db-attribute and db-entity from precedent specifications, a pr-item can be determined straightaway if it forms a valid key or not. Similarly, its specified format can be checked out conveniently. This illustrates the effects of optimum validation considerations.

As it is, the third block of INTER-COMP-DESCRIPTION comes after the MAPPING block instead of the DECLARATION block for similar reasons.

8.2.3.3 <u>INTER-COMP-DESCRIPTION</u>

The INTER-COMP-DESCRIPTION block defines the construction stage of the mapping process that corresponds to the analysis carried out in Section 7 in the earlier chapter. It specifies the association of each component to its immediate parent in terms of db-relationships. It also represents the point in the specification which relates to specific end user required semantics.

The syntax of this block is as follows :

INTER-COMP-DESCRIPTION.

$$\text{COMP} \quad \text{cname} \quad \left[\text{RELATES-TO} \begin{Bmatrix} \text{prname} \\ \text{cname} \end{Bmatrix} \quad \text{WHERE } * \quad \right]$$

* same as in Section 8.2.3.2

It must be pointed out that the linkages as defined by the WHERE clause here, will already be validated at the MAPPING block and any inconsistencies or ambiguaties detected. With acyclic schemas, the association between the various components, or more precisely, their anchors, is non-ambiguous and deterministic. In other words, there exists only one valid db-relationship connecting them in the database schema -- the semantics of the perceived record, as such, conveyed to all end users is bound by this constraint. In the latter case of ambiguaties as arising in cyclic schemas, the choice of which linkage is to be used in line with the desired semantics, can be confirmed at this point.

Conceivably, in cases of non-ambiguity, the control system can deduce the association using the anchor i.e. base information derived from the MAPPING block, without further external specification from the EUA. In short, the efforts made here again (in such circumstances) seem to be unnecessary. However, the goal here is a generalised facility and although such a deductive capability is possible, we opt not to do so.

Note too the irrelevance of this block in the case of a simple pr since it does not contain any components. Consequently, the INTER-COMP-DESCRIPTION block is omitted altogether for simple prs.

Finally, corresponding to the example specification of the DECLARATION block on page 137, the specification of the INTER-COMP-DESCRIPTION is thus :

```
COMP  Appmt    RELATES-TO  Patient   WHERE LINK-IS .......
COMP  Doctor   RELATES-TO  Appmt     WHERE LINK-IS .......
COMP  Test     RELATES-TO  Patient   WHERE LINK-IS .......
```

Again note that all the cnames and the prname will be displayed to the EUA in the interactive specification session.

Some general comments about the overall design of the language can now be best summed up in the following section.

8.2.4 Some criticisms

The above presentation has concentrated chiefly on the functionalities required of the language; there is no consideration of interfacing to a DBMS. Obviously, in practice, there needs to be some means of specifying which database schema the perceived record is derived from. An appropriate clause is thus proposed as follows, which can be inserted at the beginning of the DECLARATION block :

```
SOURCE  DB-SCHEMA  IS  schema-name
```

If the db-schema is Codasyl-based, then the db-relship used in the specification will be a Codasyl set-name. With relational schemas, it will be a JOIN predicate.

On a broader context, the language syntax is unnecessarily complex and redundant (at some places). The OR-FROM clause provides one such example of the latter. Another excessively expanded syntax is that pertaining to the WHERE LINK-IS option using INTERMEDIATE entities. These remarks are made in view of the fact that the EUA is not a casual user. The syntax could certainly be improved to a more succinct form.

8.3 Translatability of language constructs to transform model data structures

In order to show that the language proposed is adequate for its purpose, there must exist some means to translate the language declarations into the equivalent data structures in the transform model and in the external schema. Precisely, there must be some means of establishing a value for each attribute in all the data objects involved. The remainder of this section demonstrates the feasibility of such an exercise.

It is useful first to identify three groups of attributes. Where an attribute type is present in more than one entity type, association to a specific entity type is denoted by prefixing the attribute name by the entity name, such as PR-ITEM.cname. The three groups are as follows :

i)those whose values are externally provided by the user. These include : pr-name, COMP.c-name, pr-itemname, pri-role, pri-format, c-desc, db-ename, soptype, pred, linfo, ioptype, opexp, anchor and x-elem-name.

ii)those whose values are deduced from the input values or from external information. These are : pc-name, level-no, anchor, PRITEM.c-name.

iii)those whose values are generated internally. These include both the identifying attributes of X-id and Y-id.

Fig. 8.3 shows the correspondence of data values specified in the language structures to those as required for the complete definition of the various mapping and perceived record objects.

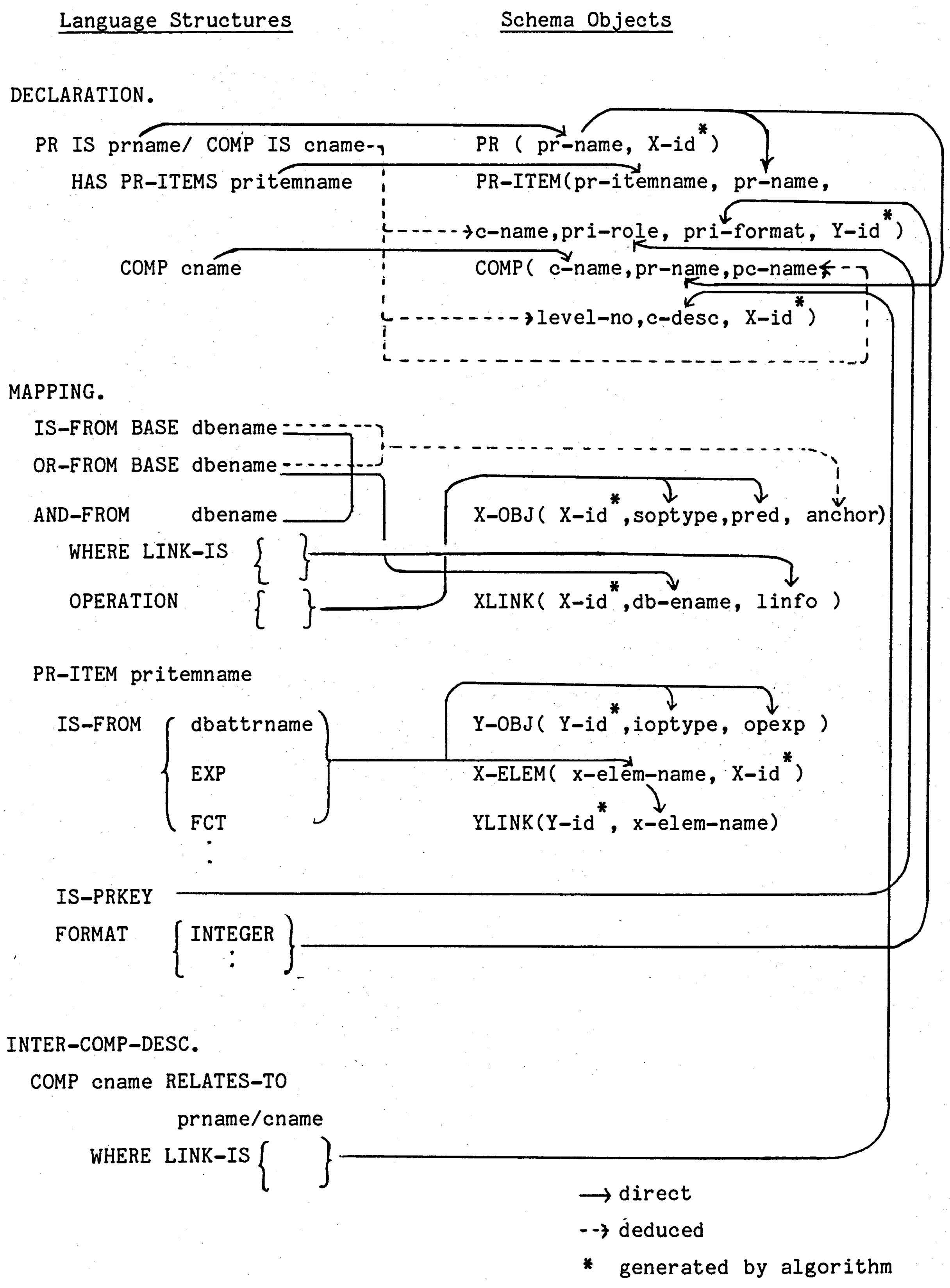


fig. 8.3 Translating Language Structures to Schema Objects

The direct correspondence of values in the fig. 8.3 is self-explanatory; as such, they will not be described further. Some explanations are necessary for the deduced values. The algorithms for obtaining X-id and Y-id values will also be discussed.

Values that must be derived are those of the attributes : PR-ITEM.c-name, pr-name, level-no and anchor. Both PR-ITEM.c-name and pc-name are deduced from the context of the specification of the DECLARATION block. The c-name for a pr-item belonging directly to a perceived record entity, and the pc-name of an immediate component of a perceived record, will have null value; otherwise, the values are determined from the c-name (supplied externally) as indicated by the dotted arrow.

The value for level-no is obtained as follows :

- o the level-no corresponding to cname specified in a PR DECLARATION block i.e. all the first level components, is system-assigned integer value 1.
- o for subsequent cname specified in the associated COMP DECLARATION blocks, the level-no of its parent(pc-name) is first checked.
- o the parent component level-no is then increased by 1 to give the required value of level-no for the component.

The mapping of anchor value is not all that straightforward from the specification. As noted in the previous chapter, when a merging transform is defined, the anchor will have a default value of 'OPT'. In other general circumstances, the value will just be the db-ename of the BASE entity as entered by the EUA.

<u>Generation of X-id and Y-id</u>

The important property of X-id and Y-id is that both must be able to uniquely identify the respective entities that they belong to. As such, they do not have to be meaningful. Consequently, both X-id and Y-id can be numbers that are generated from a code generator. Such generators would commonly be available on computer installations and therefore could be used for this purpose -- they are usually pseudo-random number generators based on linear congruent calculations [Grogono 79]. There is also no reason why X-id and Y-id thus generated cannot have the same value occurrence since they in fact belong to different entity types.

For instance, the number 12 may be both an X-id value and Y-id value pertaining to the same or different perceived record types.

Alternatively, the numbers can be generated simply by a step function which increases by 1 each time. Depending on the size of the database and the scale of perceived record derivations, the values generated in this way can theoretically grow infinitely large. For our small database example, we adopt this latter strategy.

We can now define X-id and Y-id to consist of three numeric fields. Any non 3-digit value for X-id and Y-id will be represented with leading zeros in the appropriate fields, e.g. 006.

8.4 Status of implementation

The interface system software is written in PASCAL and runs on a DEC-20 computer. The database it presently builds onto is a relational one which stores information about a hypothetical University environment. Appendix A describes this University database. The reason why the Hospital database, referred to in all previous chapters was not used for the experimentation was that tests on PRIMC started early -- a simple database example that can be readily adapted was needed to get it going. It was found that later in illustrating the various complicated aspects of mapping that the Hospital database seems to provide the kinds of realistic cases required.

Data dictionary facilities are not available at the time of implementation and the necessary features have to be explicitly coded. As already noted in Section 8.1, the resultant data descriptions from a PRIMC session are held in simple data files.

The interface so far :

- accepts the definition of a perceived record with a given level of nested components.
- supports a minimal set of transformations : direct value mapping, use of simple arithmetic expression, use of summarizing functions. Others listed in Chapter 7 have yet to be added.

The actual interactive syntax does not strictly adhere to that presented in this chapter; the modifications however are only syntactic and trivial - they do not change the underlying meaning. A guide to the dialogue with PRIMC is given in Appendix B. Example runs of the interface are provided in Appendix C.

Chapter 9 CONCLUDING REMARKS, FURTHER WORK AND APPLICATIONS

This chapter summarises the thesis and examines its significance in contributing towards database research in general, and the particular role of its application. Suggestions for specific applications of the work are also discussed.

9.1 Summary and conclusions

In the past, query languages have placed undesirable restrictions on end users in requiring familiarity with the underlying database structural constraints in order to perform effective data retrievals. The mainstream of current approaches to solving this problem lies in natural query language processing and querying by attribute naming (as in universal relation systems).

The approach in this thesis towards 'database structure independent' data retrievals is by means of high-level view support in the form of perceived records. An end user accessing data via a perceived record will only need to use simple direct retrieval commands, e.g.

GET perceived-record-name, followed possibly with a qualifying selection expression to specify the desired selection of data value occurrences.

The main contribution of this thesis lies in the study and analysis of end user view definitions that are independent of a DBMS or logical data model. The significance of a generalised view definition facility is in supporting uniform data retrieval operations to end users regardless of the underlying database model. In order to understand the requirements of mapping perceived record views, it was first necessary to develop a framework for categorising the mapping features and criteria in general, and the classes of data transformations that are possible. Altogether, five mapping features were identified : data structure support, naming, format, selection and identification of data values for a view occurrence structure. Depending on the types of views (structures) supported at the external schema, the whole set or a subset of these features may be found in a given mapping facility. A facility for mapping perceived records was found to require all five features for supporting transformations of the view structures from the database.

Two main types of data transformations were categorised in the framework : data structure transformation and data item transformation. The former was classified into concatenating, merging, subsetting, summarizing and transposing transforms. The latter was divided into two main types : conventional capabilities (possibly in the form of macros) supported by most systems to obtain derived data values, and semantic-motivated mappings. The second category is analogous to the properties of semantic data models; its importance and usefulness lie in enhancing the semantic expressiveness of data in views mapped from classical data models (relational, network or hierarchical).

The framework also discussed the properties and requirements of a mapping definition language in terms of its syntactic structure and its degree of procedurality. It was recognised that an important and ultimate issue in view mappings, namely that of ensuring the correctness of syntactically valid view occurrences (with respect to end user expectations), has yet to be resolved formally. A pragmatic approach proposed in this thesis to alleviate this problem is by means of interactively monitoring and checking the expected semantics with that in the database during a view specification.

As a specific demonstration of using the concepts and requirements, the mapping of perceived record from recursive database structures was analysed. It was shown that (end users') data retrievals from complex recursive structures are greatly simplified by the use of perceived record type of view structures. Much more significantly, using the concepts of view transformations, it was shown that the underlying essential semantics could be preserved and the information represented explicitly in perceived records that end users can perceive directly. However, it was found that the semantics of inter-dependencies between data occurrences at different recursive levels cannot be fully abstracted. This is due partly to the inherent nature of the problem and partly to the linear repeating representation adopted for views defined by the base entity concept. The formulations in recursive applications provide the logical basis for defining the semantics of necessary mapping operators. Applicability of the analysis and its usefulness in practice, was illustrated using commonly-encountered recursive examples such as BOM and Ancestor-Desendent relationships in a family-tree.

Following on, it was shown that the body of concepts and requirements could be identified in existing database systems. The framework thus serves as a means of evaluating, and provides a common basis for comparing, the mapping capabilities of practical systems. To this end, the ideas were carried out on two specific systems : System R and the Codasyl EUFC mapping proposals. A chief finding common to both was that (given that they are based on classical structured data models), both support very limited form of semantic-motivated mappings that can allow end users to perceive the underlying data objects in different role types. It is here that the kinds of semantic-motivated mappings developed in the mapping framework could be shown to be appropriately applied to these systems in order to specify the semantics of such data transformations and the corresponding operators. As a further suggestion, it would be most interesting to see how view mappings based on a semantic data model, such as in McLeod [78], would compare using this framework.

Apart from this, the comparison also revealed a general need for better-controlled specification of view mappings. By removing the freedom of end users to define views directly from database objects (in other words, leaving such tasks to the DBA or EUA), the probability of incorrect data retrievals, though not eliminated, can be reduced. Perhaps by far the most significant outcome of the evaluation was that it provided conclusive evidence for the requirement of structured view representation as advocated in the perceived record approach. More specifically, it showed that the work on perceived record mappings could be adapted to a data-processing environment such as using 'office-forms'; the extra facility required would be to map the basic perceived record structure to the visual layout of a 'form' (e.g. geometric layout facilities in the Codasyl EUFC proposals).

In the second part of the thesis then, three possible ways of specifying the perceived record model in terms of the component construct were described. The method finally adopted, integrates as part of the data model, the flexibility for an end user to perceive the base entity of the view directly in terms of pr-items (rather than necessarily via a component). Based on the flexible description of the perceived record model, two kinds of perceived record views were distinguished. These are : a simple pr representing a flat view consisting of pr-items, and a complex pr representing a structured view, consisting of possibly repeating components. A significant advantage with this model is that a perceived record, whether simple or complex, can be uniformly described in terms of one kind of data construct - the component. This property was exploited in designing a mechanism for mapping perceived records.

Due to the inherent complexity of generalising mappings for both simple and complex prs, the mechanism proposed uses an intermediary transform model. This first breaks down the transformation characteristics into a common intermediate structure called X-OBJECT which contains 'pre-processed' data items of a component, from which a perceived record can be composed. These data items may be further operated on to produce the final required pr-items in the perceived record. The concept embodied in X-object, is powerful — it does not merely establish straightforward correspondences of data structures, but in a derived form (specified by means of a structure transform), also represents a source of new semantics for view definition. In this latter form, it could be regarded as a newly-derived 'view' object and hence in a way, provide an extra dimension for directly defining a perceived record with derived semantics. In general circumstances, such a requirement will be achieved in two stages -- the derived semantics will be held in pre-defined views; new views can then be built on top of the existing views (as in System R).

Using the framework developed earlier on, three levels of mapping information are captured in the canonical transform model : that of data structure transformation, data item transformation and integrity requirements of perceived record occurrences arising from interactions of transformations at the first two mentioned levels. The application of the transform model for mapping perceived records can be used in a practical database system which has appropriate supporting software to translate the perceived record definitions into corresponding transform model object descriptions. Such descriptions will need to be held in a data dictionary to facilitate efficient validations and the subsequent retrievals of required perceived record occurrences.

A significant advantage of the transform model is the economy and flexibility in deriving many diferent perceived record types from within a single database. The economy is that the (one) specification of a component type derivation, i.e. an X-object description, may be (re)used in many perceived record definitions. The flexibility lies in the various possible ways of constructing a perceived record using different building blocks of X-objects. Its disadvantages are the extra consistency checks that need to be exercised, and the unnecessary procedures that are required for simple direct mappings (the latter being a penalty of a generalised facility).

The development of an underlying framework of mapping requirements and concepts, and the canonical transform model culminated in the design of a mapping interface facility (PRIMC) for defining perceived records. Central to this is the design of a mapping definition language. PRIMC essentially provides a general testbed for evaluating the feasibility of the mapping concepts and that of various consistency checks and their effects in practical circumstances. A perhaps more specific role is to demonstrate the application of the transform model mapping mechanism, and to consider carefully the kinds of software support that will be required in an actual working system.

It was found that the initial motivation for developing a prescriptive control system in order to minimise materialisation of unexpected end-user semantics, has a second pay-off. By carefully considering the sequence of the mapping definition language structures, the number of both semantic and syntactic validations can be reduced during the interactive specification. Similar checks that may necessarily have to be repeated at different sections of the specification need only be performed once at the very first occurrence. This, in a way, helps to overcome the major drawback imposed by the transform model mechanism. Testings with example runs of PRIMC showed that the proposal is feasible with a small database. The results were encouraging in confirming the effectiveness of optimising consistency checks using an interactively-controlled approach towards view specification. However, larger databases will need to be tested to realistically and conclusively ascertain the full benefits.

Notably, the need to predefine perceived record views (by the EUA via PRIMC) is in a sense contradicting the objective of mapping views for ad-hoc end users whose requests are often ill-defined. This contradiction arises from a paradox : on the one hand, it is desired that ad-hoc end users can define their own views; on the other hand, in order to distance them from the low level of abstractions of data in the database, someone else, in this case the EUA, is required to carry out the mapping task instead. A second reason why pre-defined views may be desirable is that it offers some kind of protection against unwary incorrect data retrievals (which often crops up when the end user needs to define more complex views, such as those from recursive structures). There is however no reason why it should not be considered that end users be allowed to define their own perceived records based on existing components. This will require careful considerations of the possibly new kinds of consistency checks that must be exercised.

Right at the beginning, it was assumed that query languages are used for data retrievals only. Consequently, perceived records considered throughout this work do not support updates. This may seem rather restrictive in a real world situation because there are occassions whereby end users may wish to delete, insert or alter certain information -- for instance, a doctor may wish to update the drug-code for a particular patient's treatment. We suggest that in such circumstances, update facilities can be provided and controlled by using an application program.

9.2 Areas of further work

Two main lines of work can be pursued : small size tasks that can be achieved quickly, and those relating to applications of this research in practical database systems that require deeper research.

9.2.1 Immediate extensions to the existing facility

At present, PRIMC provides only very basic mapping definition capabilities. A number of facilities could be added onto the existing ones to provide a more complete and sophisticated system.

(1) The set of transformation functions defined in Chapter 7 have not all been fully implemented. A first step could be to augment the remaining ones.

(2) In a second step, operators for the transformations of recursive database structures could be defined. So far, only the logical framework for achieving such mappings has been explored.

(3) The current syntax of the perceived record mapping definition language support only derivations of perceived records from database objects. For fuller exploitation of the potentiality and usefulness of the canonical transform model, the capability should be expanded to allow derivation in terms of existing perceived record objects. The EUA should then be able to define a new perceived record type comprising appropriate component types already held in the data dictionary, which will be validated against the corresponding X-objects.

(4) Another worthwhile implementation may be that of a mapping processor facility on top of PRIMC. This will provide a self-contained package for defining and retrieving perceived records.

(5) The present version of PRIMC uses only very basic and simple files for storing descriptions of the data objects. A larger and more realistic trial of PRIMC, as for example for an actual implementation on a practical system, would need more sophisticated data dictionary facilities.

9.2.2 Applications of this research

(1) Implementation on practical DBMS --

The experimentation of PRIMC so far has been with a relational database only. The ultimate objective of interfacing PRIMC to a DBMS in a realistic situation, such as System R or Ingres, or a commercially available Codasyl-based DBMS requires additional facilities. One of these, namely, data dictionary facilities, has just been mentioned in the above section.

The MAPPING block in the present version of PRIMC has database linkage clauses specific for relational predicates. For Codasyl and other types of DBMS, this will need to be made more general-purpose. A flag could be programmed into PRIMC to set up the appropriate type of linkage clauses during a perceived record definition for a given type of DBMS. In particular, with Codasyl-typed systems, additional capabilities need to be defined in order to extract information embedded in the Codasyl set (as described in Chapter 7). These functions should form part of the repertoire of mapping functions already available in the mapping definition language.

The ideal implementation should be accompanied with a mapping processor facility. There are of course interesting considerations of efficiency in such implementations and the problems of resolving the best strategy.

(2) Application in a multi-database system --

The research in this thesis has assumed throughout a single database environment. In practice, there may often be more than one database supported within one system. For instance, a hospital database, health-care database and perhaps a school database may be held in the same DBMS. It is desirable that one could issue a multi-database query request for data, for example, one might wish to find out what health-care centres that hospital doctor attends besides his hospital duties. Hence, it is such a context of requirements, that the work on mapping perceived records can be applied to a multi-database system. In general, the databases supported will be homogeneous i.e. based on the same logical data model.

It shall be assumed that in a multi-database system, descriptions of all the schema objects in the various databases are held in a central repository such as a data dictionary. The role of a data dictionary is especially important for the installation of PRIMC on top of a multi-database system to enable efficient cross-checkings of data consistency and semantic integrity between the databases. The unique identification of all perceived record objects and transform model objects in a single database environment, will now include the specific identifier of the respective databases. (A possible candidate for such a database identifier could be database-name).

A common feature of multi-database access is data incompatibility, especially that due to variance in the scale or unit that the same piece of data may be assigned in different databases. Given that the consistency constraints between the data objects are satisfied (e.g. ensuring that they belong to the same domain), these cases could be handled in a manner consistent with the present version of PRIMC — for instance, with unit or scale difference examples, transformation rules could be coded in a table that could be applied to the appropriate data objects in order to map to a standard scale or unit in the perceived record.

(3) <u>View mappings in a distributed database environment</u> --

The successful implementation of the work defined in (1) and (2) above should provide the pre-requisites for applying the mapping ideas to a distributed database environment, that may be homogeneous or heterogeneous. This generally can mean two things, which are as follows :

(i) The end user may retrieve data from any one database at any one time that is supported by a local DBMS connected in a distributed network system. For instance, one may wish to access another database held physically elsewhere about say, housing information.

(ii) More significantly, the end user may wish to retrieve data from more than one database within a single query at any one time, i.e. a multi-database request. In practice, these DBMSs are commonly heterogeneous. Such situations may arise when there are pre-existing databases containing information of a similar nature that can be meaningfully associated. For example, information about different universities held in local databases at each site may be integrated together. A possible query may be to find lecturers from all universities who teach Mathematics. Another possible query may be to find out the marking scheme of all the universities where each university may have different standards for the scheme. This latter example illustrates yet another case of dealing with incompatible data.

One more issue of incompatibility that must be considered carefully is that of different data structure representations (pertaining to the heterogeneous context). To illustrate, consider the two database schemas below :

a) a network schema

$$\begin{tikzcd}
\text{Student(S-no,S-name)} \arrow[dr] & & \text{Course(C-no, C-name)} \arrow[dl] \\
& \text{Mark(Tma1, Tma2, Tma3)} &
\end{tikzcd}$$

b) a relational schema

Mark(Ms-no, Mc-no, Tma-no, Tma-result)

In order to define a perceived record,

Student-Grade(S-no, C-no, S-name, (Tma-no, Mark)) merged from the above schematic data, the source entities for the merging transform must be in compatible form. This raises issues that need to be researched further into, such as whether transformations can be defined on a mixture of database types and X-object types, the use of a global data dictionary, besides other implementational and efficiency considerations. An outline of a possible mapping architecture that uses the canonical transform model as a basis for mapping perceived records might be as follows :

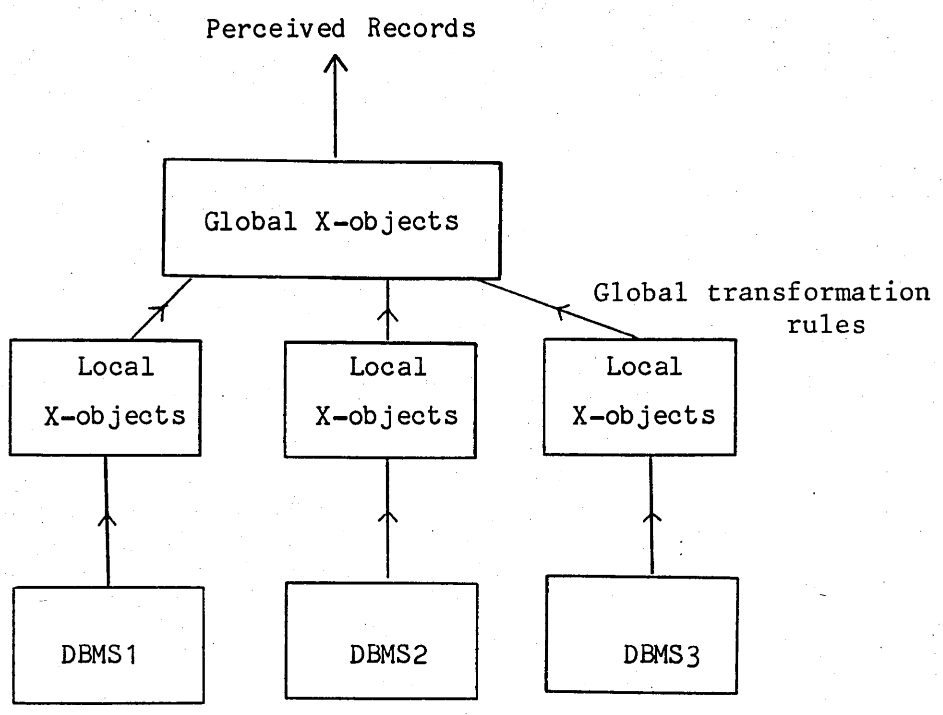


fig. 9.1 Outline of a possible perceived record mapping architecture in a distributed database system

9.3 Final appraisal

Following from the above discussions on the practical applications of the perceived record approach, it is appropriate to end the thesis by looking at its overall impact in resolving some fundamental issues in database querying.

The direction adopted is one based on the structure and semantics of a view to simplify end-user retrievals. The end user is presented with a restricted but precise set of semantics contained within a unit structure which represents all the information that is required.

So far, the other main two approaches, namely that of natural language querying and attribute-based querying represent in contrast, a non-structured approach. These claimed that by not imposing any form of data structure, users can then express their queries more easily and readily according to their real-world requirements. This although true to a certain extent, can have dire consequences in producing incorrect data retrievals. In both approaches, there is little information about the underlying database semantics communicated to the end user. As already stated in Chapter 1, it was indeed found that the structuring of data does help the user to understand its meaning and that natural language querying is not that natural as it sounds. Similarly, in attribute-based querying such as the universal relation approach, there is the same element of inadequate provision of semantics that is essential for proper and effective retrievals. Again, the partial solution to this lies in enhancing the user's perception by appropriately structuring data presented to users.

In such approaches, with a lack of semantic constraints to guide users to interprete data themselves, the system attempts to interpret and deduce any semantic ambiguaties that may arise in users' queries. Although some of such systems do provide a mechanism for interacting with the user, or allowing user-specification of actual semantics required, this process can only be effective if the user can be fully aware of the semantic constraints of the underlying data.

In the light of the kinds of specific query applications described in the previous section, it is particularly noted that both these approaches cannot be applied to the last two areas without aggravating the kinds of problems mentioned above. The universal relation approach though, by definition can support homogeneous multi-database retrieval as the user is given the illusion that he or she is accessing a single relation only.

In summarizing, the perceived-record approach in overcoming the problems, offers a simple and more effective means of data retrieval that can be employed in specific database application areas as compared to natural-language and attribute-based querying.

Appendix A The University Database

The University database model is extracted from OU [81]. The information described in the database relates to Students, Tutors and Courses.

Each Student has a Tutor-Counsellor; each Tutor-Counsellor counsels many Students.
Each Student enrols in many Courses; each Course registers many Students.
Each Student might do many Previous-course.

The corresponding relational logical schema is as follows:

Student (S-no, S-prefix, S-name, S-year, Counsellor)
Tutor-c (Tc-no, Tc-prefix, Tc-name, Marker)
Course (C-no, C-title)
Enrolment (Es-no, Ec-no, Tutor)
Mark (Ms-no, Mc-no, Tma-no, Tma-result)
Prev-Course (Ps-no, Pc-no, Pc-year, Pc-grade)
Inactive (Is-no, Is-prefix, Is-name, Is-year)

Appendix B The Interactive Dialogue

Notes :

- o User inputs are prompted by generic data names within angle brackets e.g. < dbattrname >. Hence, all data names immediately following the prompts are typed in by the user, except in the cases of :
  - i) a blank - this signifies that the user does not want the option
  - ii) a '?' - the system provides a Help facility to the user in the MAPPING and INTER-COMP-DESCP block to guide the user as to what database types are valid. To invoke this, the user types a '?'. The system also prescribes the correct database connections in response to a '?'.

- o The system responds with error messages if any semantic inconsistencies or mismatches of data names are detected.

- o At the end of each successful specification for a perceived record, the entire perceived record structure is displayed to the user. Names of perceived records and components always begin with a capital letter; pritem-names, on the other hand are in lower-case letters. Currently, the system depends on the user to input the data names in the correct lower/upper cases.

Appendix C

Example runs of PRIMC to define perceived record views based on the University database

These examples show the dialogues by which a user declares and specifies the mappings for:

(1) A simple pr with a pr-item summarised from a different db-entity type.

(2) A complex pr with one component.

(3) A complex pr with nested components.

(4) A pr merged from different db-entity types.

(5) A complex pr containing a transformed component merged from two different db-entity types.

The following consistency checks have been implemented :-

o Check 1 - This checks that the input data name is a valid domain value. It is essentially a syntactic check on the value.

o Check 2 - This is used in the MAPPING block. It checks that within component* structure mapping, in the case of multiple source db-entities, there exists a valid semantic link between each non-base source and the base entity of the component* structure. This is invoked when the db-ename value is input. Effectively the check ensures the correct identification of db-entity occurrences in a concatenating transform.

o Check 3 - This again is applied in the MAPPING block. It checks that the base entity of every component entity mapping has valid links to the base entity of the associated perceived record entity mapping. If none is found, the user will be asked if the connection is via an intermediate db-entity.

Contd.

- Check 4 - This occurs in the INTER-COMP-DESCRIPTION block. It checks that the input db-link between every component and perceived record base, which specifies the required end-user semantics is correct. This effectively registers the higher, second-level of occurrence identification. Note that by virtue of Check 3 applied earlier in the MAPPING block, there must exist at least one explicitly stored relationship i.e. there is one specific set of semantics that can be associated to the perceived record.

- Check 5 - Given that Checks 2 to 4 are not violated, i.e. there exists a valid relationship, this then checks that the corresponding link-attributes input are correct.

- Check 6 - This checks that for a mrgset operation, the input data names belong to the same class of entity.

Annotations in italics have been inserted alongside the respective listings in order to guide the user in understanding the dialogues better.

Example Run 1

```
     ** A SEMANTIC CONTROLLER TO DEFINE PERCEIVED RECORD VIEWS**
            ** If option is NOT required, press RETURN
               Otherwise input data names as required **
-------------------------------------------------------------------------------

DECLARATION.
------------

PERCEIVED RECORD IS < prname > Student
     HAS PRITEM(S) <pritemnames> s-num,s-name,s-year,no-course-enrolled
     HAS COMPONENT(S) < cnames >



MAPPING.
--------

 PERCEIVED RECORD Student
         IS-FROM BASE-ENTITY < dbename > student
         OR-FROM BASE-ENTITY < dbename >

         AND-FROM NONBASE-ENTITY < dbename > course
....no relationship exists between base and nonbase     Consistency
entities....Is link from an intermediate (nsource)   Check 2 invoked;
entity ?....Type Y/N >>  Y
         VIA NON-SOURCE-ENTITY <dbename> enrolment
             WHERE LINK IS-FROM student
                                <dbattrname> s-no
             WHERE LINK IS-FROM enrolment
           (NOTE : two link attributes are required)
                                <dbattrname> es-no
                                <dbattrname> ec-no
             WHERE LINK IS-FROM course
                                <dbattrname> c-no
.... to end derivation, press RETURN ....
         AND-FROM NONBASE-ENTITY < dbename >

         HAS < OPERATION >

     PR-ITEM s-num
         IS-FROM  < dbattrname >s-no

     PR-ITEM s-name
         IS-FROM  < dbattrname >s-name

     PR-ITEM s-year
         IS-FROM  < dbattrname >s-year                   Note that no INTER-
                                                         COMP-DESC generated
     PR-ITEM no-course-enrolled                          for simple pr.
         IS-FROM  < dbattrname >
         IS-FROM  < arithmetic-exp. *
         IS-FROM <funct.(dbattrname)>count(c-no)


     **** PERCEIVED RECORD DEFINITION IS SUCCESSFUL ****

                                                       Display of simple pr
NEW PERCEIVED RECORD DEFINED :                         Student

 Student( s-num, s-name, s-year, no-course-enrolled)
```

Example Run 2

```
     ** A SEMANTIC CONTROLLER TO DEFINE PERCEIVED RECORD VIEWS**
           ** If option is NOT required, press RETURN
              Otherwise input data names as required **
-------------------------------------------------------------------------------

DECLARATION.
------------

PERCEIVED RECORD IS < prname > Tutor-student
     HAS PRITEM(S) <pritemnames> tc-no,tc-name
     HAS COMPONENT(S) < cnames > Student

COMPONENT IS Student
     HAS PRITEM(S) <pritemnames> s-num,s-nameyear
     HAS COMPONENT(S) < cnames >
```

Only one component declared

```
MAPPING.
--------

 PERCEIVED RECORD Tutor-student
         IS-FROM BASE-ENTITY < dbename > tutor-c
         OR-FROM BASE-ENTITY < dbename >

         AND-FROM NONBASE-ENTITY < dbename >

         HAS < OPERATION >

     PR-ITEM tc-no
         IS-FROM  < dbattrname >tc-no

     PR-ITEM tc-name
         IS-FROM  < dbattrname >tc-name


 COMPONENT Student
         IS-FROM BASE-ENTITY < dbename > student
         OR-FROM BASE-ENTITY < dbename >

         AND-FROM NONBASE-ENTITY < dbename >

         HAS < OPERATION >
```

Mapping for component begins

```
     PR-ITEM s-num
         IS-FROM  < dbattrname >s-num
....dbattribute is not valid ...try again
                < dbattrname > ? one of the following:
 counsellor   s-name    s-no  s-prefixs-year
                < dbattrname > s-no
```

Consistency Check 1 invoked

Input of '?' prompts system to give choice of valid names

Contd.

Contd. Example Run 2

```
INTER-COMP-DESCP.
-----------------

COMPONENT Student RELATES-TO BASE  Tutor-student
             WHERE LINK IS-FROM tutor-c
                                <dbattrname> tc-no
             WHERE LINK IS-FROM student
                                <dbattrname> s-no
? Does not match directory or user name -
                                <dbattrname> ? "counsellor"
                                <dbattrname> counsellor


    **** PERCEIVED RECORD DEFINITION IS SUCCESSFUL ****

NEW PERCEIVED RECORD DEFINED :

 Tutor-student( tc-no, tc-name, Student( s-num, s-nameyear))
```

Automatic generation of data names and pr structure

Consistency Check 5 invoked System prescibes correct name on user prompt of '?'

Example Run 3

```
      ** A SEMANTIC CONTROLLER TO DEFINE PERCEIVED RECORD VIEWS**
             ** If option is NOT required, press RETURN
                Otherwise input data names as required **
--------------------------------------------------------------------------

DECLARATION.
------------

PERCEIVED RECORD IS < prname > Student-details
     HAS PRITEM(S) <pritemnames> s-no,s-name
     HAS COMPONENT(S) < cnames > Course,Tutor-c

COMPONENT IS Course
     HAS PRITEM(S) <pritemnames> c-no,c-name
     HAS COMPONENT(S) < cnames > Mark

COMPONENT IS Mark
     HAS PRITEM(S) <pritemnames> tma-no,tma-result
     HAS COMPONENT(S) < cnames >

COMPONENT IS Tutor-c
     HAS PRITEM(S) <pritemnames> tc-prefix,tc-name
     HAS COMPONENT(S) < cnames >
```

Level-1 components within pr

Level-2 component embedded within component Course

No further component nested

Level-1 component Tutor-c has no nested structure

```
MAPPING.
--------

PERCEIVED RECORD Student-details
        IS-FROM BASE-ENTITY < dbename > student
        OR-FROM BASE-ENTITY < dbename >

        AND-FROM NONBASE-ENTITY < dbename >

        HAS < OPERATION >

    PR-ITEM s-no
        IS-FROM  < dbattrname >s-no

    PR-ITEM s-name
        IS-FROM  < dbattrname >s-name


COMPONENT Course
        IS-FROM BASE-ENTITY < dbename > course

         -----------Warning-------------
      there is no stored relationship between the anchor entity of this
       component to the base entity of the perceived record
...... to abort and start all over again, press CONTROL key and C key toget
     else continue input .........
```

Consistency Check 3 invoked

Contd.

```
        OR-FROM BASE-ENTITY < dbename >

        AND-FROM NONBASE-ENTITY < dbename >

        HAS < OPERATION >

    PR-ITEM c-no
        IS-FROM  < dbattrname >c-no

    PR-ITEM c-name
        IS-FROM  < dbattrname >c-title


COMPONENT Mark
        IS-FROM BASE-ENTITY < dbename > mark
        OR-FROM BASE-ENTITY < dbename >

        AND-FROM NONBASE-ENTITY < dbename >

        HAS < OPERATION >

    PR-ITEM tma-no
        IS-FROM  < dbattrname >tma-no

    PR-ITEM tma-result
        IS-FROM  < dbattrname >tma-result


COMPONENT Tutor-c
        IS-FROM BASE-ENTITY < dbename > tutor-c
        OR-FROM BASE-ENTITY < dbename >

        AND-FROM NONBASE-ENTITY < dbename >

        HAS < OPERATION >

    PR-ITEM tc-prefix
        IS-FROM  < dbattrname >tc-prefix

    PR-ITEM tc-name
        IS-FROM  < dbattrname >tc-name
```

Program has not been aborted after check 3 warning; indicates a possible valid link that user prefers

```
INTER-COMP-DESCP.
-----------------

COMPONENT Course RELATES-TO BASE  Student-details
        VIA NON-SOURCE-ENTITY <dbename> enrolment
            WHERE LINK IS-FROM student
                               <dbattrname> s-no
            WHERE LINK IS-FROM enrolment
                    (NOTE : two link attributes required)
                               <dbattrname> es-no
                               <dbattrname> ec-no
            WHERE LINK IS-FROM course
                               <dbattrname> c-no

COMPONENT Mark RELATES-TO COMPONENT  Course
            WHERE LINK IS-FROM course
                               <dbattrname> c-no
            WHERE LINK IS-FROM mark
                               <dbattrname> mc-no
```

Note: earlier in MAPPING block, system detects no direct link the source db-entities here. User now proceeds to define a required semantic linkage via an intermediate

Contd.

Contd. Example 3

```
COMPONENT Tutor-c RELATES-TO BASE  Student-details          Consistency
             WHERE LINK IS-FROM student
                                <dbattrname> s-no           Check 5 invoked
? Does not match directory or user name -
                                <dbattrname> counsellor
             WHERE LINK IS-FROM tutor-c
                                <dbattrname> tc-no


     **** PERCEIVED RECORD DEFINITION IS SUCCESSFUL ****


NEW PERCEIVED RECORD DEFINED :                              Nested structure
                                                  representation of complex pr
 Student-details( s-no, s-name, Course( c-no, c-name, Mark( tma-no, tma-result
 Tutor-c( tc-prefix, tc-name))
```

```
     ** A SEMANTIC CONTROLLER TO DEFINE PERCEIVED RECORD VIEWS**
            ** If option is NOT required, press RETURN
               Otherwise input data names as required **
-------------------------------------------------------------------------------

DECLARATION.
------------

PERCEIVED RECORD IS < prname > All-students
     HAS PRITEM(S) <pritemnames> s-no,s-prefix,s-name,year-enrolled
     HAS COMPONENT(S) < cnames >



MAPPING.
--------

 PERCEIVED RECORD All-students
        IS-FROM BASE-ENTITY < dbename > student
        OR-FROM BASE-ENTITY < dbename > inactive
        OR-FROM BASE-ENTITY < dbename >

        HAS < OPERATION >

    PR-ITEM s-no                  DERIVED-FROM-DB-ENTITY   student
        IS-FROM   < dbattrname >s-no
                                  OR-DERIVED-FROM-DB-ENTITY   inactive
        IS-FROM   < dbattrname >sis-no

    PR-ITEM s-prefix              DERIVED-FROM-DB-ENTITY   student
        IS-FROM   < dbattrname >s-prefix
                                  OR-DERIVED-FROM-DB-ENTITY   inactive
        IS-FROM   < dbattrname >is-prefix

    PR-ITEM s-name                DERIVED-FROM-DB-ENTITY   student
        IS-FROM   < dbattrname >s-name
                                  OR-DERIVED-FROM-DB-ENTITY   inactive
        IS-FROM   < dbattrname >is-name

    PR-ITEM year-enrolled         DERIVED-FROM-DB-ENTITY   student
        IS-FROM   < dbattrname >
        IS-FROM   < arithmetic-exp. =
        IS-FROM <funct.(dbattrname)>
        IS-FROM   < dbattrname >s-year
                                  OR-DERIVED-FROM-DB-ENTITY   inactive
        IS-FROM   < dbattrname >is-year


     **** PERCEIVED RECORD DEFINITION IS SUCCESSFUL ****


NEW PERCEIVED RECORD DEFINED :

 All-students( s-no, s-prefix, s-name, year-enrolled)
```

User indicates one other optional base entity for pr

&

&

&

&

& -- system automatically knows that the pr-items have optional source db-attributes

```
   ** A SEMANTIC CONTROLLER TO DEFINE PERCEIVED RECORD VIEWS**
         ** If option is NOT required, press RETURN
            Otherwise input data names as required **
------------------------------------------------------------------------

DECLARATION.
------------

PERCEIVED RECORD IS < prname > Student-course
    HAS PRITEM(S) <pritemnames> s-no,s-name,year
    HAS COMPONENT(S) < cnames > Course-attended

COMPONENT IS Course-attended
    HAS PRITEM(S) <pritemnames> c-no
    HAS COMPONENT(S) < cnames >



MAPPING.
--------

 PERCEIVED RECORD Student-course
        IS-FROM BASE-ENTITY < dbename > student
        OR-FROM BASE-ENTITY < dbename >

        AND-FROM NONBASE-ENTITY < dbename >

        HAS < OPERATION >

     PR-ITEM s-no
        IS-FROM  < dbattrname >s-no

     PR-ITEM s-name
        IS-FROM  < dbattrname >s-name

     PR-ITEM year
        IS-FROM  < dbattrname >s-yaear


COMPONENT Course-attended
        IS-FROM BASE-ENTITY < dbename > course
```

Consistency Check 3 invoked

```
         -----------Warning-------------
      there is no stored relationship between the anchor entity of this
       component to the base entity of the perceived record
...... to abort and start all over again, press CONTROL key and C key together
     else continue input ........


        OR-FROM BASE-ENTITY < dbename > prev-course
        OR-FROM BASE-ENTITY < dbename >
```

User indicates that the component has optional bases

```
        HAS < OPERATION >

    PR-ITEM c-no                 DERIVED-FROM-DB-ENTITY   course
        IS-FROM  < dbattrname >c-no
                                 OR-DERIVED-FROM-DB-ENTITY   prev-course
        IS-FROM  < dbattrname >pc-year
   ....domain type is not correct.... try again ....
                < dbattrname > ? one of the following:
pc-grade   pc-nopc-year    ps-no
                < dbattrname > pc-no
```

Consistency Check 6 invoked System prompts user with list

Contd.

```
INTER-COMP-DESCP.
-----------------

COMPONENT Course-attended RELATES-TO BASE  Student-course
        VIA NON-SOURCE-ENTITY <dbename> enrolment
            WHERE LINK IS-FROM student
                               <dbattrname> studs-no
            WHERE LINK IS-FROM enrolment
                   (NOTE : two link attributes required)
                               <dbattrname> es-no
                               <dbattrname> ec-no
            WHERE LINK IS-FROM course
                               <dbattrname> c-no
```

User defines desired link

```
    **** PERCEIVED RECORD DEFINITION IS SUCCESSFUL ****


NEW PERCEIVED RECORD DEFINED :

 Student-course( s-no, s-name, year, Course-attended( c-no))




    ** A SEMANTIC CONTROLLER TO DEFINE PERCEIVED RECORD VIEWS**
          ** If option is NOT required, press RETURN
             Otherwise input data names as required **
--------------------------------------------------------------------------

DECLARATION.
------------

^C
^C
@logoport
Open or close? close
```

REFERENCES

Aho, A. V. and Kernighan, B. W. [80] Researcher user/ava/q/README, (1980).

Anderson, B. F. and Dale, A. G. [77] A proposed language for database transformations, in Data Models and Database Systems, Proc. of the joint US-USSR Seminar, pp. 95-119 (Moscow Nov. 14-23, 1977).

ANSI [75] ANSI : Interim Report ANSI/X3/SPARC Study Group on Database Management Systems (1975).

Atkinson, M. P., Bailey, P. J., Chisholm, K. J., Cockshott, P. W. and Morrison, R. [83] An approach to persistent programming, the Computer Journal, 26/4, pp. 360-365 (Nov. 1983)

Astrahan, M. M., Blasgen, N. W., Chamberlain, D. D., Eswaran, K. P., Gray, J. N., Griffiths, P. P., King, W. F., Lorie, R. A., McJones, P. R., Mahl, J. W., Putzolu, G. R., Traiger, I. L., Wade, B. W. and Watson, V. [76] System R: Relational approach to database management, ACM Transactions on Database Systems 1/2, pp. 97-137 (1976).

Atzeni, P. and Parker, D. S. [82] Assumptions in relational database theory in Proc. of ACM Symposium on Principles of Database Systems, pp. 1-9 (Los Angeles, Calif., March 29-31, 1982).

BCS QL [81] Query languages : a unified approach, British Computer Society monograph in Informatics, (ed. Samet, P. A.) Heydon & son Ltd. (1981).

Bell, D.M.R. [81] IDBACS : Interactive DataBase Administrator Control System, IUCC Bulletin Vol. 3, pp. 66-70 (1981).

Biller, H. [79] On the equivalence of database schemas - a semantic approach to data translation, Information Systems 4/1, pp. 35-47 (1979).

Biskup, J., and Bruggemann, H. [83] Universal relation views: a pragmatic approach, in Proc. 9th Int. Conf. on Very Large Data Bases, pp. 206-218 (Florence, Italy, 1983).

Bobrow, R. J. [71] An Experimental Data Management System, in Courant Compute Science Symposia 6, " Data Base Systems ", pp. 123-141 (New York 1971).

Bracchi, G., Fedili, A., Paolini, P. [74] A multilevel relational data model for data base management system, in Data Base Management ed. Klimbie, J. W., Koffeman, pp. 211-213 North-Holland, Amsterdam (1974).

Bracchi, G., Paolini, P., Pelagatti, G. [76] Binary logical associations in data modelling, in Modelling in Database Management Systems, ed. Nijssen, G.M. , IFIP (1976).

Brodie, M. L. [78] Specification and verification of data base semantic integrity, Ph. D. thesis, Depart. of Computer Science, Univ. of Toronto, Toronto (1978).

Brodie, M.L. [79] Association : A database abstraction for semantic modelling, in Entity-Relationship-Approach to Information Modelling and Analysis (ed. Chen, P. P.) pp. 583 - 608 (1979).

Brodie, M. L. [81] On modelling behavioural semantics of databases, in Proc. of 7th, Int. Conf. on Very Large Data Bases, (Cannes, France, 1981).

Carlson, C. R. and Kaplan, R. S. [76] A Generalised Access Path model and its application to a relational database system, in Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 143-154 (1976).

Chamberlain, D. D., Gray, J. N., Traiger, I. L. [75] Views, authorization and locking in a relational data base system, in Proc. AFIPS NCC Conference, Vol.44, pp. 425-430 (1975).

Chen, P. P. [76] The Entity-Relationship model - toward a unified view of data, ACM Transactions on Database Systems, 1/1, pp. 9-36 (1976).

Claybrook, B. G. [82] A specification method for specifying data and procedural abstractions, IEEE Transactions on Software Engineering 8/5, pp. 449-459 (Sept. 1982).

Claybrook, B.G. [83] Defining database views as data abstractions in EXT-PASCAL, Information Systems 8/3, pp. 165-176 (1983).

Clemons, E. K. [76a] Design of an external schema facility for a relational database, Decision Sciences Working Paper 76-11-03, Univ. of Pennsylvania, Philadelphia (1976).

Clemons, E. K. [76b] Design of a user interface for a relational database, Ph. D. Dissertation, School of Operational Research and Industrial Engineering, Cornell University, Ithaca, N. Y. (1976).

Clemons, E. K. [78] The external schema and Codasyl in Proc. of 4th Int. Conf. on Very Large Databases, pp.130 (Berlin, Germany, 1978).

Clemons, E. K. [79a] An external schema facility for Codasyl 1978, in Proc. of 5th Int. Conf. on Very Large Data Bases, pp. 119-128 (Rio de Janeiro, Brazil, 1979).

Clemons, E. K. [79b] Design of a prototype ANSI/SPARC three schema database system, in Proc. of NCC AFIPS conf., Vol. 48 pp. 689-696 (New York 1979).

Clemons, E. K. [81] Design of an external schema facility to define and process recursive structures, ACM Transactions on Database Systems, 6/2, pp.295-311 (1981).

Codasyl [69] Codasyl 1969 Data Base Task Group Report, ACM, New York (Oct. 1969).

Codasyl [78] Codasyl DDLC Journal of Development (1978).

Codasyl EUFTG [75] A Progress Report on the Activities of the CODASYL End User Facility Task Group, (June 1975).

Codasyl EUFC [83] Codasyl End User Facilities Committee Journal of Development (1983).

Codd, E. F. [70] A relational model for large shared data banks, Communications of ACM 13/6, pp. 377-387 (1970).

Codd, E. F. [72] Relational completeness of data base sublanguages, in Data Base Systems, Courant Computer Science Symposium. 6th., ed. Rustin, R., pp.65-98, Prentice-Hall, Englewood Cliffs, NJ (1972).

Codd, E. F. [74] Seven steps to rendezvous with the casual user, in Database Management (ed. by Klimbie,J. and Koffeman, K.) , North Holland, Amsterdam (1974).

Codd, E. F. [79] Extending the database relational model to capture more meaning, ACM Transactions on Database Systems 4, pp. 397-434 (1979).

Dale, A. G., and Dale, N. B. [76] Schema and occurrence structure transformation in hierarchical systems, in Proc. of ACM SIGMOD Int. Conf. on Management of Data pp. 157-168 (New York 1976).

Dale, A. G., and Dale N. B. [77] Main schema - external schema interaction in hierarchically organised databases, in Proc. ACM SIGMOD Conf. on the Management of Data, pp. 102-110 (Toronto 1977).

Dale, A. G. and Lowenthal, E. I. [77] End-user interfaces for database management systems, in ANSI/SPARC DBMS Model (ed.) Jardine, D. A., pp. 81-99, (North Holland 1977).

Date, C. J. [75] An architecture for high-level language database extensions, in Proc. of ACM SIGMOD Int. Conf. on Management of Data pp. 101-122 (1975).

Date, C. J. [77, 81] An introduction to database systems (2nd., 3rd. edition) Addison-Wesley, (1981).

Dayal, U., Bernstein, P. A. [78] On the updatability of relational views, in Proc. of 4th. International Conference on Very Large Data Bases, pp. 368-377 (1978).

Esslemont, P. and Gray, P. [82] The performance of a relational interface to a Codasyl database, in Proc. of 2nd British National Conf. on Databases pp. 21-34 (1982).

Fagin, R. [78] On an authorization mechanism, ACM Transactions on Database Systems, 3/3, pp. 310-319 (1978).

Fagin, R., Mendelson, A. O. and Ullmann, J. D. [80] A simplified universal relation assumption and its properties, IBM Tech. Report RJ2900, San Jose (November 1980).

Fallahzadeh, H. and Cousins, T. R. [83] Relational views and operators for network - structured databases, in Proc. of the 16th. Annual Hawaii Int. Conf. on System Sciences pp. 170-183 (1983).

Goldman, J. [79] Automated generation of relational schemas for Codasyl networks, Sperry Research Center, Sudbury, Ma., (March 1979).

Griffiths, P., Wade, B. [76] An authorization mechanism for a relational database system, ACM Transactions on Database Systems, 8/1, pp.1-14 (1976).

Grogono, P. [79] Programming in PASCAL, (3rd edition) Addison-Wesley, (1979).

Hammer, M. M. and Mcleod, D. [78] The semantic database model : a modelling mechanism for database applications, in Proc. of ACM SIGMOD Int. Conf. on the Management of Data, (Austin,Texas 1978).

Hammer, M. M. and Mcleod, D. [81] Database description with SDM: A Semantic Data Model, ACM Transactions on Database Systems, 6/3, pp. 351-86 (Sept. 1981).

Harris, L. [78] User oriented database query with the ROBOT natural language query system, Int. Journal of Man-Machine Studies 9, pp. 697-713 (1978).

Housel, B. C. and Shu, N. C. [76] A high-level DML for hierarchical data structures, Proc. of ACM SIGMOD Conf. on data: Abstraction, Definition and Structure, pp. 155-169 (1976).

Jones, C. B. [80] Software development: A Rigorous approach, Prentice Hall (1980).

Kalinenchenko, L. A. [76] Relational-network data structure mapping, in Modelling in Database Management Systems (ed) Mijssen, G. M. North-Holland, IFIP, pp. 303-309 (1976).

Kay, M. H. [75] An assessment of the Codasyl DDL for use with a relational subschema, in Database Description, (ed) Douque, B. C. M., Nijssen, G. M. pp. 199-212 (1975).

Kameny, I. [78] EUFID : the end-user friendly interface to data management systems. Proc. of Very Large Databases, (West Berlin, Germany, Sept. 1978).

Kent, W. [81] Consequences of assuming a universal relation, ACM Transactions on Database Systems 6/4, pp. 539-556 (Dec. 1981).

Klug, A. [78] Theory of database mappings, Ph. D. thesis, Depart. of Computer Science, Univ. of Toronto, Toronto (1978).

Klug, A. [81] Multiple view, multiple data model support in the CHEOPS database management system, Univ. of Wisconsin, T. R. No. 418 (Jan. 1981).

Klug, A. and Tschritzis, D. C. [77] Multiple view support within the ANSI/SPARC framework, in Proc. of 3rd. Int. Conf. on Very Large Data Bases, pp. 447-488 (Tokyo, Japan 1977).

Korth, H. F. [81] System/u: a progress report, in Proc. XP/2 Conf., (June 1981).

Korth, H. F. and Ullman, J. D. System/u: a database system based on the universal relation assumption, in Proc. XPI Conf., Stonybrook, New York (June 1980).

Kreps, P. [80] Relativism and views in a conceptual database model, ACM Proceddings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Colarado, pp. 141-143 (1980).

Kuck, S. M., McNabb, D. A., Rice, S. V. and Sagiv, Y. [80] The Parafrase database user's manual, Computer Science Technical Report 80-1046, Univ. of Illinois (Dec. 1980).

Kuck, S. M. and Sagiv, Y. [82] A universal relation database system implemented via the network model, in Proc. of ACM. Symp. on Principles of Database Systems, pp. 147-157 (March 1982).

Lacroix, M. and Pirotte, A. [83] Comparison of database interfaces for application programming, Information Systems 8/3 pp. 217-229 (1983).

Leavenworth, B. [81] Database views using data abstraction, IBM. Res. Report RC 8722 (1981).

Leavenworth, B. M., Sammet , J. E. [74] An overview of non-procedural languages , ACM SIGPLAN Notices 9, no. 4, pp. 1-12 (April, 1974).

Lien, Y. E. [81] Hierarchical schemata for relational databases, ACM Transactions on Database Systems, 6/1, pp. 48-69 (March 1981).

Lien, Y. E. [82] On the equivalence of database models, Journal of ACM 29/2, pp. 33-362 (April 1982).

Lisboa, E. T. [79] Une proposition de correspondance entre interfaces externes heterogenes et interface conceptuelle d'une architecture de bases de donnes reparties, These de Docteur Ingenicur, Universite Paris 6, (Juin 1979).

Liskov, B. H. et al [77] Abstraction mechanisms in CLU, Communications of ACM 20/8, pp. 564-576 (Aug. 1977).

Lockemann, P., Mayr, H. C., Weil, W. H. and Wohlleber, W. H. [79] Data abstractions for database systems, ACM Transactions on Database Systems 4/1, pp. 60-75 (1979).

Maier, D. [80] Discarding the universal instance assumption: preliminary results, XP1 Workshop on Relational Database Theory, (June-July 1980).

Maier, D. [83a] Windows on the World, in Proc. of ACM SIGMOD Record Vol., 13, Part 4, pp. 68-78 (1983).

Maier, D. [83b] The theory of relational databases, Pitman (1983).

Maier, D., Rozenshtein, D., Salveter, S. C., Stein, J. and Warren, D. S. [82] Toward logical data independence: A relational query language without relations, in Proc. of ACM SIGMOD Conf. pp. 51-60 (June 1982).

Manola, F., Pirotte, A. [81] Codasyl Query Language Flat (CQLF) specifications and capabilities description , NBS Report, National Bureau of Standards, Washington, D. C. (available as CCA Technical Report CCA-81-10), (July 1981).

Manola, F., Pirotte, A. [82] CQLF - A query language for Codasyl-type databases, in International Conference on Management of Data, ACM SIGMOD pp. 207-212 (June 2-4, 1982).

McLeod. D. [78] A semantic data base model and its associated structured user, Massachusetts Institute of Technology, Cambridge Lab. for Computer Science (1978).

McLeod, D. [82] A prescriptive database interface methodology for end-users, ACM Transactions on Database Systems 7/3, pp. 253-262 (1982).

Mehl, J. W. and Wang, C. P. [74] A study of order transformations of hierarchical structures in IMS databases, IBM Res. REport RJ1359 (21046), San Joe, (March 1974).

Mylopoulos, J., Borgida, A., Cohen, P., Roussopoulos, N., Tsotsos, J. and Wong, H. [76] TORUS : a step towards bridging the gap between databses and the casual user, Information Systems 2/2, pp.49-64 (1976).

Navathi, S. B. [80] An intuitive approach to normalize network structured data, Proc. of Int. Conf. on Very Large Data Bases, pp. 350-358 (1980).

Navathi, S. B. and Fry, J. P. [76] Restructuring for large databases: Three levels of abstraction, ACM Transactions on Database Systems 1/2, pp. (June 1976).

Nijssen, G. M. [76] A gross architecture for the next generation database management systems, in Modelling in Database management systems (ed.) Nijssen, G. M. pp. 1-25, IFIP North-Holland (1976).

Nijssen, G. M. [77] On the gross architecture for the next generation database management system, in Information Processing, (ed.) Gilchrist, B. pp. 327-335 IFIP (1977).

OU [80] Open University, M352 Computer-based Information Systems; Block II, Open University Press (1980).

OU [81] Open University, M352 Computer-based Information Systems; Activity Booklet III, Open University Press (1981).

Pelagatti, G., Paolini, P. and Bracchi, G. [78] Mapping external views to a common data model, Information Systems 3/2, pp. 141-151 (1978).

Reisner, P. [81] Human factors studies of database query languages : a survey and assessment, ACM Computing Survey, 13/1, pp. 13-31 (1981).

Rowe, L. A. and Shoens, K. A. [79] Data abstraction, views and updates in RIGEL, in Proc. of ACM SIGMOD Conf. pp. 71-81 (1979).

Rozenshtein, D. [82] Query and authorization of updates in the association object data model, Comptuer Science Technical Report 82-040, SUNY at Stony Brook (May 1982).

Schmidt, J. W. [77] Some high level language constructs for data of type relation, in ACM Transactions on Database Systems 2/3, pp. 247 - 261 (September 1977).

Schmidt, M. A. and Swenson, J. R. [75] On the semantics of the relational model, in Proc. ACM SIGMOD Conf. pp. 211-233 (San Jose 1975).

Selinger, P. [80] Authorization and views, in Distributed Data Bases ed. Draffan, I. W., Poole, F. pp. 233-247 (1980).

Shipman, D. W. [81] The functional data model and the data language DAPLEX ACM Transcations on Database Systems 6/1, pp. 140-173 (March 1981).

Smith, J. M., Bernstein, P. A., Dayal, U., Goodman, N., Landers, T., Lin, K. W. T. and Wong, E. [81] Multibase - integrating heterogeneous distributed database systems, in Proc. of AFIPS Conf. Vol. 50 pp. 487-499 (1981).

Smith, J. M. and Smith, D. C. P. [77a] Database Abstractions: Aggregation and generalization, ACM Transcations on Database Systems 2/2 pp. 105-133, (June 1977).

Smith, J. M. and Smith, D. C. P. [77b] Database Abstractions: Aggregation, ACM Transactions on Database Systems 20/6 pp. 405-413 (June 1977).

Shneiderman, B. [78] Improving the human factors aspect of database interactions, ACM Transactions on Database Systems 3/4, pp. 417-439 (1978).

Spaccapietra, S. [80] Heterogenous database distribution, in Distributed Data Bases (ed.) Draffan, I. W. and Pools, F. pp. 174-193, Cambridge University Press (1980).

Stein, J. [83] Data definition and update in the association-object data model, Ph. D. dissertation in preparation, SUNY at Stony Brook (1983).

Stonebraker, M. [75] Implementation of integrity constraints and views by query modification in Proc. of ACM SIGMOD Workshop on management of data, pp. 65-78, San Jose, Calif. (May 1975).

Stonebraker, M., Wong, E. [74] Access control in a relational database management system by query modification, Electronics Research Lab. Memorandum : ERL-M438, UC Berkeley, California (May 1974).

Swartwout, D. [77] An Acess Path Specification Language for restructuring network databases, in Proc. ACM-SIGMOD Conference on the Management of Data, pp. 88-101 (Toronto, 1977).

Tagg, R. M. [83] Interfacing a query language to a Codasyl DBMS in ACM SIGMOD Record (USA) 13/3 pp. 46 - 64 (April 1983).

Tsichritzis, D. C., Lochovsky, F. H. [82] Data models, Prentice-Hall (1982).

Ullman, J. D. [82] The U. R. strikes back, in Proc. of ACM Symp. on Principles of Database Systems pp. 10-22 (March 1982).

Ullman, J. D. [83] Universal relation interfaces for database systems, Informatin Processing 83, (ed.) Mason, R. E. A., (North-Holland, 1983).

Vassiliou, Y. and Lochovsky, F. H. [80] DBMS transaction translation Computer Software and Applications Conf., Chicago, pp. 89-96 (Oct. 1980).

Waltz, D. [78] An English language question answering system, in Communications of ACM 21/7, pp. 526-539 (1978).

Wasserman, A. I. [79a] A software engineering view of database management, in Proc. of 9th Int. Conf. on Very Large Data Bases, pp. 23-35 (Belgium 1979).

Wasserman, A. I. [79b] The data management facilities of PLAIN, in Proc. of ACM SIGMOD Conf., pp. 60-70 (1979).

Weber, H. [79] A software engineering view of database systems, in Proc. of 9th Int. Conf. on Very Large Data Bases, pp. 36-51 (Belgium 1979).

Zaniolo, C. [79a] Multimodel external schemas for Codasyl Database Management Systems, in Data Base Architecture (ed.) Bracchi, and Nijssen, G. M. pp. 171-190 (North-Holland, IFIP 1979).

Zaniolo, C. [79b] Design of relational views over network schemas, in Proc. ACM SIGMOD Conf. pp. 179-190, (Boston 1979).

Zloof, M. M. [75] Query By Example , AFIPS Conference Proceedings, National Computer Conference 44, pp. 431-438 (1975).

Zloof, M. M. [76] Query By Example : operations on the transitive closure , Researh Report RC 5526, IBM Res. Centre, Yorktown Heights (new York Oct. 1976).

Zloof, M. M. [77] Query-By-Example : a data base language, IBM Systems Journal, no.4, pp. 324-343 (1977), Vol. 16.