Environments for viewpoint representations

Nigel Stanger and Richard Pascoe
Department of Information Science, University of Otago
Dunedin, New Zealand
Email: nigel.stanger@stonebow.otago.ac.nz

[*To appear in Proceedings of the Fifth European Conference on Information Systems (ECIS '97), June 18–21 1997, Cork, Ireland*]

Environments for viewpoint representations

Abstract

Modelling the structure of data is an important part of any system analysis project. One problem that can arise is that there may be many differing viewpoints among the various groups that are involved in a project. Each of these viewpoints describes a perspective on the phenomenon being modelled. In this paper, we focus on the representation of developer viewpoints, and in particular on how multiple viewpoint representations may be used for database design. We examine the issues that arise when transforming between different viewpoint representations, and describe an architecture for implementing a database design environment based on these concepts.

1. Introduction

Modelling the structure of data is an important part of any system analysis project. In this paper, the modelling of data is discussed using the concepts of perspectives, viewpoints, representations, techniques and schemes, as shown in Figure 1, and suggested by Finkelstein (1989), Easterbrook (1991a) and Darke and Shanks (1995).

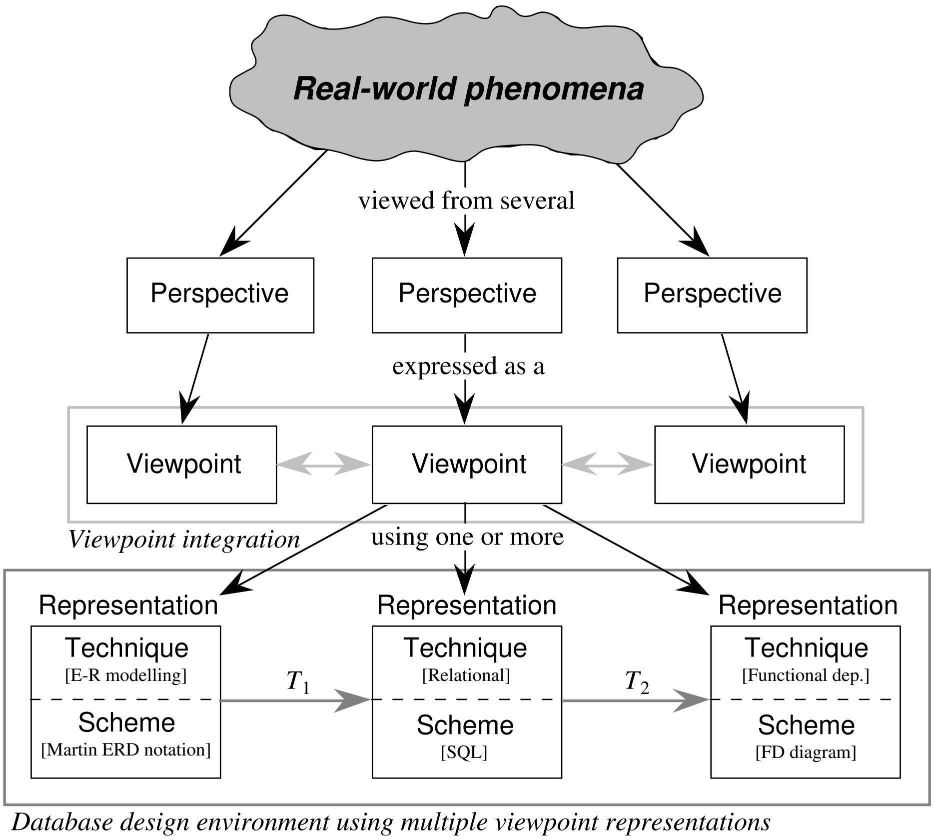

Figure 1. Perspectives, viewpoints and representations.

A *perspective* is a description of some real-world phenomenon that has internal consistency and a specified focus (Easterbrook 1991a). During the requirements definition phase of systems analysis, developers often encounter many different perspectives on the problem being modelled. Perspectives may overlap, or even conflict with each other. Part of the process of database design is deciding how to deal with these multiple perspectives. This is an active area of research and has been discussed by several authors (Leite and Freeman 1991; Easterbrook, Finkelstein et al. 1994; Kotonya and Sommerville 1996).

A *viewpoint* is a formatted expression of a perspective (Finkelstein, Goedicke et al. 1989). Darke and Shanks (1995) define two main types of viewpoint, *user viewpoints* and *developer viewpoints*. These viewpoints may be described using various *representations*, each of which comprises a *technique* expressed in some notation or *scheme*. A technique may have one or more associated schemes, but each combination of a technique and a scheme to describe a viewpoint forms a distinct representation. For example, the relational model (RM) is a technique, with SQL and QUEL being two possible schemes, but the combinations RM + SQL and RM + QUEL form two distinct representations. Figure 1 does not show techniques with multiple schemes for the sake of clarity.

Darke and Shanks (1995) grouped representations into three categories:

- *informal* representations, consisting of unstructured descriptions, often expressed using a natural language;

- s*emi-formal* representations, consisting of structured descriptions. Examples include entity-relationship modelling and data flow diagrams; and

- *formal* representations, consisting of structured descriptions and a set of operators for processing these descriptions. Examples include the relational model (Codd 1970) and logic-based languages.

Unlike informal representations, which are often ill-defined, inconsistent and incomplete, semi-formal and formal representations are well-defined, consistent and unambiguous. A key feature of formal representations that is lacking in semi-formal representations is the inclusion of operators which allow us to make assertions about the viewpoints being described. User viewpoints are typically defined using informal representations, whereas developer viewpoints are typically defined using more formal representations.

In general, no single representation will be adequate to fully describe all types of viewpoint, and indeed, the current plethora of modelling techniques suggests that a single representation is inadequate to fully describe even a *single* viewpoint. The approach suggested here is to use multiple representations to describe a particular viewpoint. This is because the use of many representations, each with their slightly different constructs, allows a more complete description of a viewpoint to be formed than that formed using a single representation.

Thus, we can say that a viewpoint is specified by a set of *descriptions*, each using some kind of representation to describe either the whole viewpoint, or some subset of the viewpoint. These descriptions may be distinct from each other, or they may overlap. For example, Figure 2 shows a developer viewpoint of a simplified used-car dealership, specified by the union of four descriptions: D_1, an entity-relationship diagram (ERD); D_2, a functional dependency diagram (FDD); D_3, an SQL schema; and D_4, a data flow diagram (DFD). Some of these descriptions overlap considerably (for example, D_1 and D_3), whereas others overlap to a lesser extent (for example, D_1 and D_4). Each description uses a different representation.

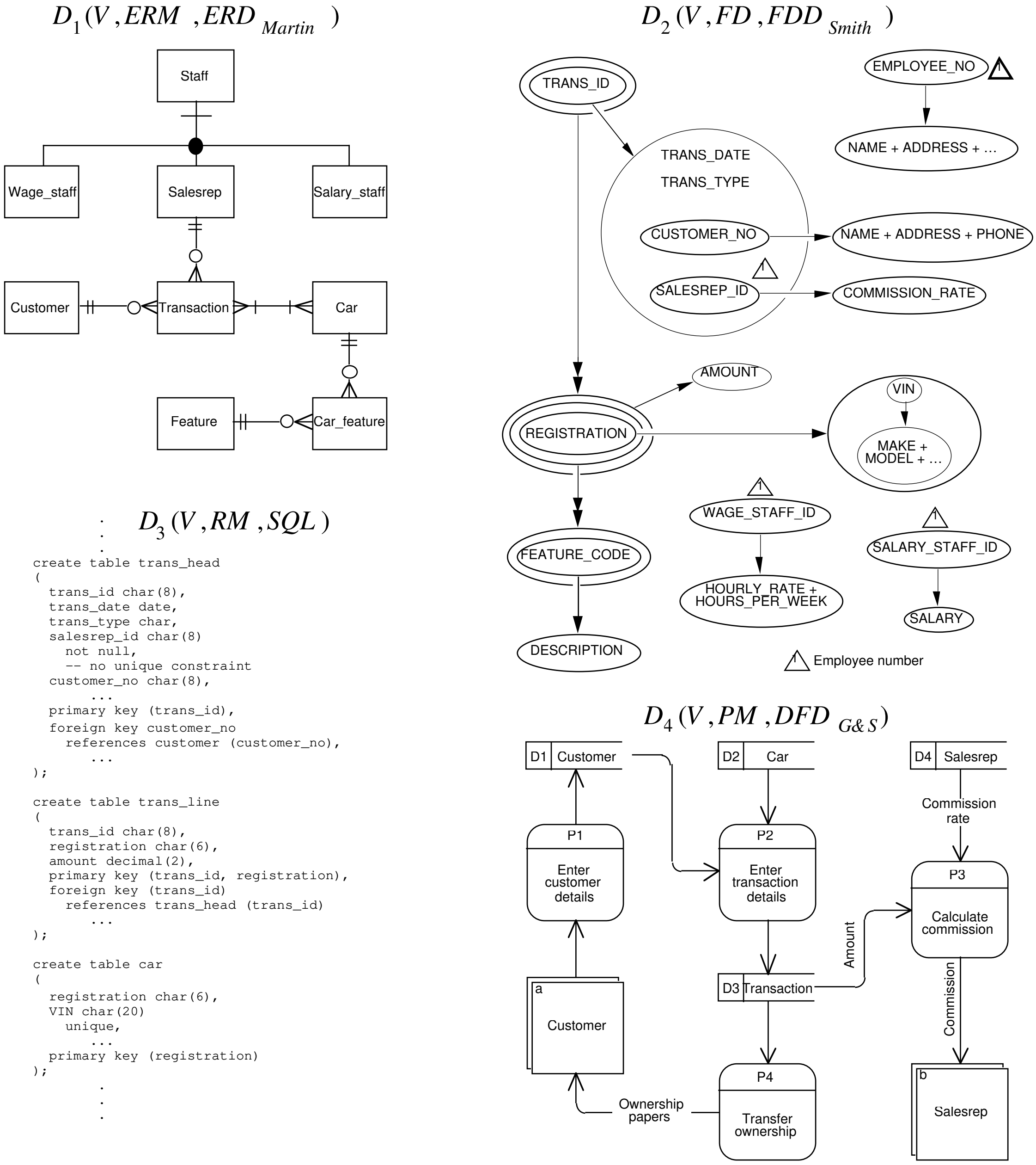

Figure 2. Four descriptions of the same phenomenon.

We introduce the notation *D(V, T, S)* to mean a description *D* of viewpoint *V* comprises constructs defined by technique *T*, and expressed using scheme *S*. For example, in Figure 2, description D_1, denoted by $D_1(V, ERM, ERD_{Martin})$, is represented using entity-relationship modelling (ERM) and expressed as an ERD in Martin notation (Evergreen Software Tools 1995); $D_2(V, FD, FDD_{Smith})$ is represented using functional dependencies (FD) and expressed as an FDD using Smith's notation (Smith 1985); $D_3(V, RM, SQL)$ is represented using the relational model (RM) and expressed in SQL; and $D_4(V, PM, DFD_{G\&S})$ is represented using process modelling (PM) and expressed as a DFD in Gane & Sarson notation.

Representations may differ in both the technique and scheme used, or they may differ only in the scheme. D_1, D_2, D_3 and D_4 all differ in both the technique and the scheme used. If we were to add $D_5(V, RM, QUEL)$ to the viewpoint in Figure 2, we would have two descriptions that use the same technique (D_4 and D_5), but different schemes (QUEL instead of SQL).

Much of the work into viewpoints has been from a software engineering perspective. Our focus, however, is on database design. In this paper, we examine how to facilitate the database design process when using multiple representations within a particular viewpoint, which is an area that has only just begun to receive attention (Darke and Shanks 1995). In other words, given descriptions $D_i(V, T_i, S_i)$, V remains constant for all values of i, otherwise we will enter the area of viewpoint integration.

If we consider a database design environment that allows the use of multiple viewpoint representations (indicated in Figure 1 by the dark grey box), then a useful feature for such an environment would be to allow designers to shift from one representation to another at will, so that they may more fully describe the problem at hand. Ideally, the environment should handle as much of this shifting process as possible, leaving the designer to "fill in the gaps". What we therefore need is some means of transforming viewpoint descriptions from one representation to another. In considering this problem, we restrict ourselves to developer viewpoints using formal and semi-formal representations. At present, informal representations are not considered because they generally lack the structure required to enable effective transformations between representations to be carried out (see further research later in this paper).

In the next section we examine the issues that arise when performing such transformations, and give examples of the problems that can arise. In the third section, we discuss an architecture for the implementation of a database design environment which facilitates the use multiple viewpoint representations. This architecture is derived from work done in the area of automatic data translation. The fourth section discusses implementation issues and directions for future research.

2. Issues

When transforming viewpoint descriptions between different representations there are three main issues to consider:

- how the actual transformation process is carried out;
- the overall integrity of the design that is produced; and
- the quality of the transformations.

The examples given in this section are based on the used-car dealership model shown in Figure 2. The examples focus on those elements which relate directly to a transaction (either a sale or a purchase), such as salesreps, customers and cars, and have been chosen to illustrate many of the "interesting" elements of the transformation in question.

2.1. The transformation process

In the introduction, we described the rationale for transforming viewpoint descriptions from one representation to another. We shall denote the transformation operator using the symbol $\rightarrow$ and describe the transforming of a description D_1 in one representation to a description D_2 in another representation as $D_1(V, T_i, S_j) \rightarrow D_2(V, T_k, S_l)$. For example, transforming a viewpoint description expressed using SQL into one using QUEL is denoted by $D_1(V, RM, SQL) \rightarrow D_2(V, RM, QUEL)$ and transforming a Chen-style ERD into a relational schema is denoted by $D_1(V, ERM, ERD_{Chen}) \rightarrow D_2(V, RM, SQL)$.

As noted earlier, a transformation may involve changing both the technique and the scheme, or just the scheme. We shall call these two types of transformation *technique transformations* and *scheme transformations* respectively. At first glance, it may appear that scheme transformations are trivial and do not need to be considered. It is important to note, however, that some schemes do not fully adhere to the technique that they express. For example, SQL does not fully adhere to the definition of the relational model (Date and Darwen 1993). In addition, some schemes may be able to express more information than others, for example, some ERD notations can express AND/OR relationships, whereas others cannot. Although we cannot disregard scheme transformations, we shall only examine two simple examples of technique transformations: one from an entity-relationship "model" to the relational model, and one from the relational model to functional dependencies.

2.1.1. *Entity-relationship → relational*

In this example, we are transforming from an entity-relationship "model" expressed in Martin ERD notation, to the relational model expressed in ANSI SQL/92, that is $D_1(V, ERM, ERD_{Martin}) \rightarrow D_2(V, RM, SQL/92)$. This transformation is illustrated by the arrow labelled T_1 in Figure 1. The ER representation is a semi-formal representation, while the relational representation is a formal one.

This transformation should be familiar to most readers, as SQL schema generation from an ERD is probably one of the most common operations performed by CASE tools. Figure 3 shows an example of such a transformation. In this case, we are transforming the entities for Salesrep and Transaction into equivalent SQL structures.

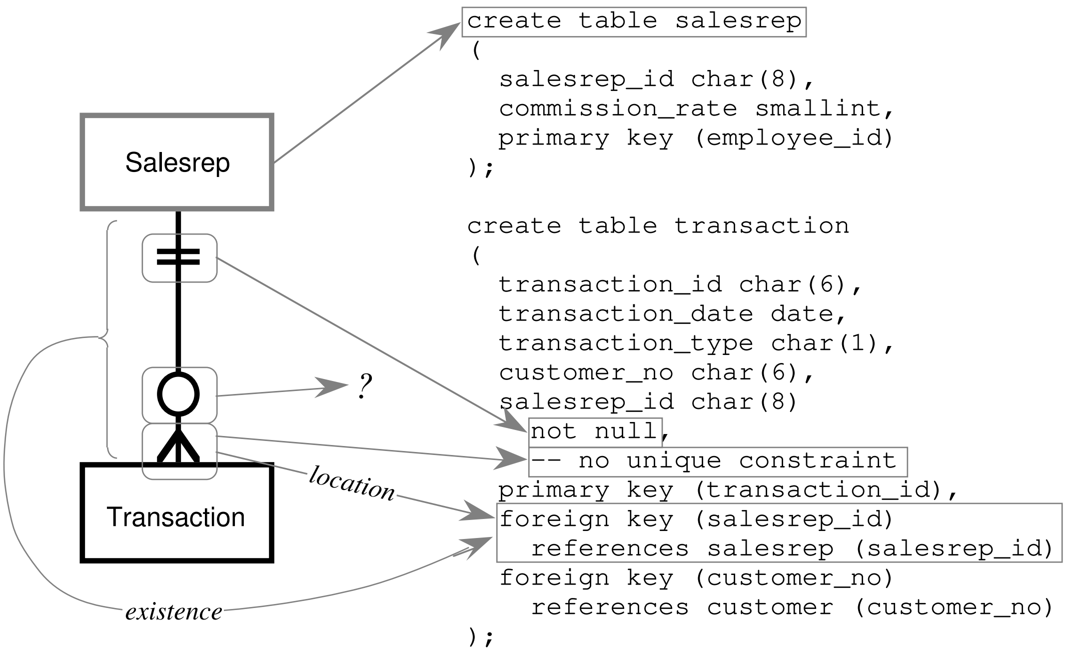

Figure 3. Transformation from an ERD to SQL.

The arrows in Figure 3 indicate the transformation of a particular ER construct into a corresponding SQL construct. For example, the relationship between the two entities indicates that there is a primary/foreign key relationship between them, and the location of the foreign key is determined by the cardinality of the relationship. In this case, the Transaction entity is on the "many" side of the relationship, so the foreign key is placed in the Transaction table definition.

There are three main points that arise from the analysis of this transformation. First, the optionality of the referenced entity (Salesrep in Figure 3) is expressed in SQL by the existence (or not) of a NOT NULL constraint on the foreign key attribute of the referencing entity (Transaction). This is an important aspect of referential integrity that is often overlooked and it is interesting to note that many CASE tools do not appear to implement this rule.

Second, a similar situation also arises with the cardinality of the referencing entity (Transaction). This is expressed in SQL by the existence (or not) of a UNIQUE constraint on the foreign key attribute of the referencing entity. Again, many CASE tools do not appear to implement this.

Third, the optionality of the referencing entity (Transaction) cannot be expressed in SQL, as there is no equivalent SQL construct. This is a limitation of SQL, however, rather than of the relational model itself — it should be possible to represent such a constraint using relational calculus or some similar scheme. In other words, this is a limitation of the particular representation that we have chosen.

The inverse transformation $D_2(V, RM, SQL/92) \rightarrow D_1(V, ERM, ERD_{Martin})$ is similar, in that the optionality of the referencing entity cannot be determined from the SQL code, whereas the first two items can be determined, *if* the SQL code has been correctly defined.

2.1.2. Relational → functional dependencies

In this example we are transforming from the relational model expressed in SQL, to functional dependencies expressed as an FDD using Smith's notation. That is $D_1(V, RM, SQL/92) \rightarrow D_2(V, FD, FDD_{Smith})$. This is illustrated by the arrow labelled T_2 in Figure 1. Both representations are formal ones. Figure 4 shows an example of such a transformation.

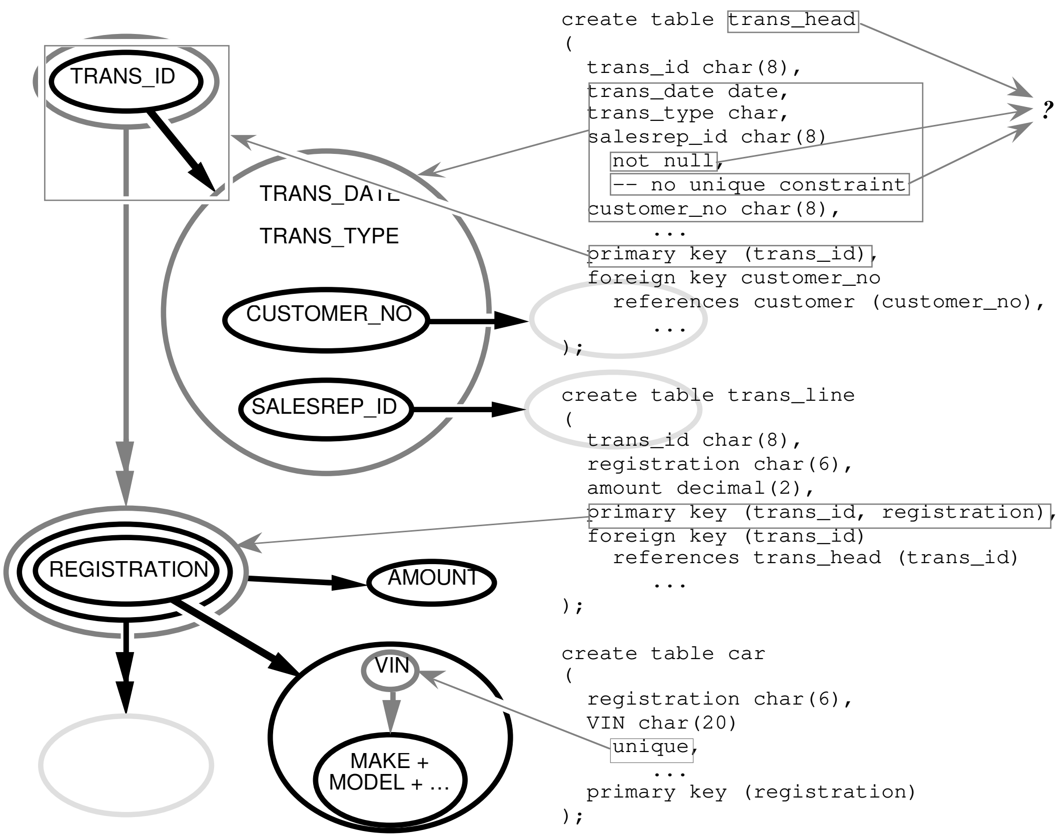

Figure 4. Transformation from SQL to an FDD.

The points to note here are:

- Functional dependencies are concerned only with the relationships between attributes, so relation names are lost in the transformation.
- "Optionality" information is lost completely, although as previously noted, SQL can only partially represent optionality.
- The "cardinality" of the referencing relation in a primary/foreign key relationship is lost.

The inverse transformation $D_2(V, FD, FDD_{Smith}) \rightarrow D_1(V, RM, SQL/92)$ is again similar: we cannot derive table names from the FDD, and there is no optionality or cardinality information to be transformed. This information must be derived from elsewhere; we shall return to this issue shortly.

2.1.3. Analysis

Our analysis has shown that there is considerable overlap between entity-relationship modelling and the relational model. If we were to use a representation that fully adhered to the relational model, then the overlap may be total. If we use SQL, however, we cannot transform the optionality of a referencing entity.

Similarly, we find that there is a lesser overlap between the relational model and functional dependencies, perhaps on the order of 60%. Because of the considerable overlap between entity-relationship modelling and the relational model, it seems reasonable to state that there is probably also about a 50–60% overlap between entity-relationship modelling and functional dependencies.

We acknowledge that these observations are rather *ad hoc*, however, and that some form of quantitative or qualitative measure is needed to determine the degree of overlap between different representations. If we can determine this overlap, we can use this as a basis for developing a database design environment that supports multiple representations.

2.2. Integrity of the design

Suppose a developer uses a single representation to describe a viewpoint (remember that we are assuming only a single viewpoint). How can they verify the integrity of this description? In the past, a design would often be developed and implemented, and only then discovered to be inadequate (Brooks 1975). One approach to solving this problem is to adopt a prototyping methodology (Sallis, Tate et al. 1995), which, although potentially time-consuming, can lessen the discontinuity between the original requirements and the final solution.

If the same developer used multiple representations instead, they could use these different descriptions to aid in testing the integrity of the overall design. For example, as shown in Figure 5(a), if we were to independently create an ERD (D_1) and an FDD (D_3) to describe the viewpoint, the integrity of the combination could be verified by transforming both D_1 and D_3 into relational form (D_2 and D_4) and determining whether D_2 and D_4 were equivalent. That is, given $D_1(V, ERM, ERD) \rightarrow D_2(V, RM, SQL)$ and $D_3(V, FD, FDD) \rightarrow D_4(V, RM, SQL)$,we need to determine whether $D_2 \equiv D_4$.

Alternatively, as shown in Figure 5(b), we could transform D_1 into a second FDD (D_5) and compare this with D_3 to determine whether D_1 and D_3 were equivalent. That is, given $D_1(V, ERM, ERD) \rightarrow D_5(V, FD, FDD)$ and $D_3(V, FD, FDD)$ we need to determine whether $D_3 \equiv D_5$.

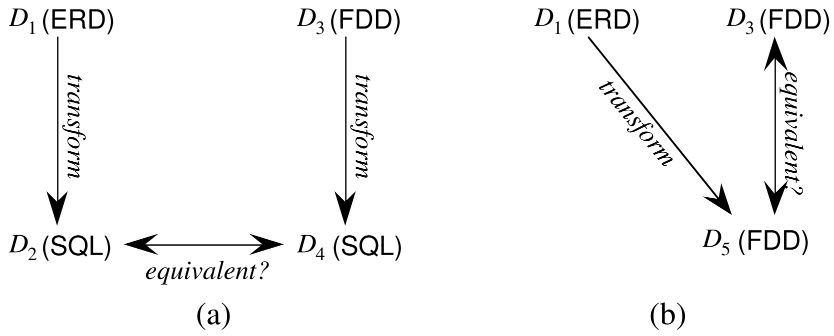

Figure 5. Integrity checking strategies using different representations.

If the required equivalence does not hold in either of the examples above, this can be caused by one of two things:

(1) D_1 and D_3's representations are irreconcilable, that is, D_1's representation expresses some information that D_3's cannot, or vice versa; or

(2) there is a viewpoint inconsistency between the two descriptions, that is, either D_1 or D_3 is inconsistent with the definition of viewpoint V.

The first case is to be expected, as different representations generally express different information. For example, functional dependencies cannot express entity names. The second case, however, may reveal an inconsistency in the design that must be addressed by correcting either D_1 or D_3. Alternatively, D_2, D_4 or D_5 may describe a new viewpoint V_2, for example $D_3(V, FD, FDD) \rightarrow D_4(V_2, RM, SQL)$. Such a situation is beyond the scope of this paper.

Thus, the use of multiple representations for a single viewpoint improves design integrity in two ways. First, different representations may be able to describe aspects of the viewpoint that other representations cannot, allowing us to more fully describe the viewpoint and produce a better database design. Second, inconsistencies in the original viewpoint may be exposed when described using a different representation.

2.3. Quality of transformations

As stated previously, different representations will most likely overlap in what they can represent, although the degree of this overlap may vary considerably. An important issue that arises when performing a transformation at either the technique or the scheme level is the loss and/or gain of information that occurs. For example, entity names are an important element of the ER approach, but they cannot be represented by functional dependencies, so this information is apparently "lost" in the transformation. The information is not truly lost, however, as it is still contained within the original ERD. Gain of information, on the other hand, is a problem that must be dealt with. For example, if we start with a set of functional dependencies and transform this into an ERD, we must generate the entity names in some way.

This is a particular problem with technique transformations, as the information represented can vary quite considerably from technique to technique, as discussed earlier. The same problem can also occur in scheme transformations, but to a much lesser extent, as the differences between schemes are generally not as radical as those between techniques.

An important task, therefore, is to determine what the overlap is between different representations, so that we can determine the information that needs to be “gained” (where applicable) when transforming between representations. This knowledge will ultimately determine the efficacy of any database design environment that may be developed.

3. Environments for viewpoint representations

Our aim is to develop a database design environment which facilitates the use of multiple representations. In this section we briefly discuss possible architectures for building such an environment.

Let us return for a moment to the notion of viewpoints. One problem that arises from the use of multiple viewpoints is that they may conflict. Traditionally, the solution to this has been to define a single “correct” viewpoint that may or may not integrate elements from the other viewpoints (Finkelstein and Sommerville 1996). This approach follows the objectivist paradigm (Klein and Hirschheim 1987), which effectively states that there is always some kind of underlying “truth” to be found when modelling reality. In other words, we can always find one “correct” solution.

In reality, however, this is just not the case. Experience has shown that there are usually several different, but equally “correct”, solutions to any but the most trivial of data modelling problems. The process of generating a single “correct” viewpoint may result in much useful information being discarded, especially when different viewpoints conflict, as there is usually some reason for such conflicts. Easterbrook (1991) refers to this problem as the “single viewpoint bias” and notes that it has been criticised by several authors (Cunningham, Finkelstein et al. 1985; Shaw and Gaines 1989). Recent work in this area has focused on negotiated resolution of conflicts between viewpoints (Easterbrook 1991b; Easterbrook and Nuseibeh 1995; Easterbrook and Nuseibeh 1996). This approach follows the subjectivist paradigm, which states that there is no underlying “truth”, only interpretations of reality.

Viewpoints are a relatively high-level concept, as indicated in Figure 1. We are dealing with a lower level, that of how viewpoints are represented, and it is interesting to note that the use of multiple representations to describe a viewpoint mirrors the use of multiple viewpoints to describe a phenomenon. This is an important point to bear in mind when considering the type of architecture to use for implementing an environment which can deal with multiple viewpoint representations.

3.1. Architectures

One possible approach is to attempt to define some kind of uniform representation (similar to an interchange format). All information is described using this representation, and it is transformed to other representations as required. The main disadvantage is that the uniform representation may become a “moving target” — as new representations are added, the uniform representation must be updated to handle them (Pascoe and Penny 1990). This can lead to a proliferation of incompatible versions that cannot communicate.

Another argument against this strategy is that it represents a kind of “representation integration” step, which is analogous to the objectivist approach of unifying all viewpoints into a single “correct” viewpoint. In other words, we are conceptually moving away from the viewpoint-based approach.

Also consider that the end result of the database design process is usually the generation of a database schema. If we are going to take our multiple representations and transform them into a single representation anyway, it seems somewhat unnecessary to introduce yet another representation into the mix, especially if the "target" representation is one of the original set of representations (for example, SQL).

A better approach for database design seems to be to perform transformations between representations as needed. The total number of transformation "engines" required increases, but adding a new representation does not result in a new, incompatible version. Also, as stated above, this approach is analogous to the way in which viewpoints are handled. It may also provide other advantages to the schema generation process which will be discussed in the next section.

Figure 6 illustrates an architecture which follows this strategy. The transformations between representations are represented by the grey arrows; T_1 and T_2 correspond to the two examples discussed earlier. Each representation is placed into its own module, which handles all the needs of that representation. This modular approach makes the environment much easier to extend.

Each module has a *technique component*, which deals with storage issues, and a *scheme component*, which deals with the user interface. Technique components can be shared across representations, as shown by the relational technique in Figure 6, which is shared across two relational representation modules, one that uses SQL and another that uses QUEL.

If such an architecture is used, we need to consider the issue of information gain/loss when transforming from one model to another. Since different representations may store different information, it would seem sensible to store representation-specific information with that representation. When additional information is required to complete a transformation, the environment must query the developer in some way to obtain that information, if it is not possible to generate it automatically.

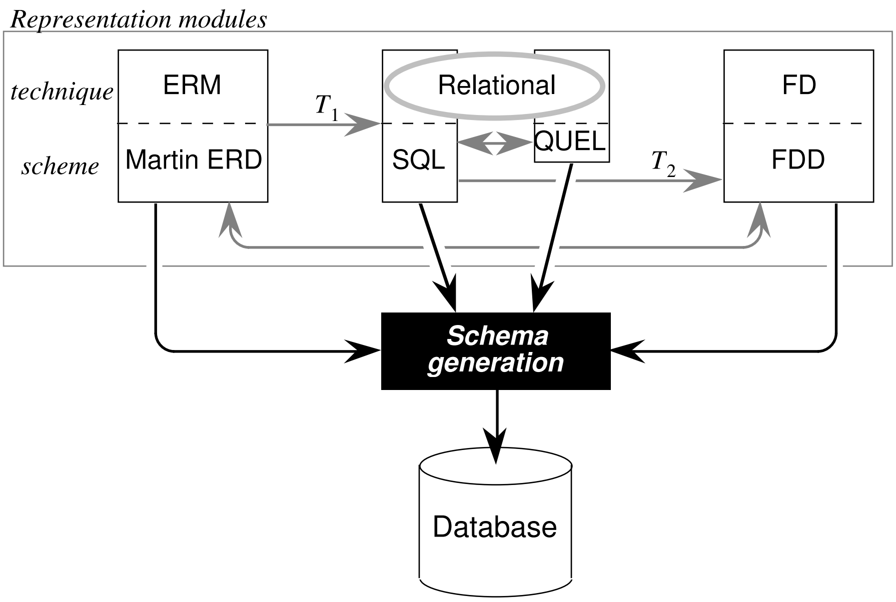

Figure 6. An architecture for a multiple representation database design environment.

4. Implementation and further research

Implementation of an environment, as described in the previous section, has not yet begun. It is planned that a preliminary implementation and results will be completed by the time of publication. In the remainder of this section, we briefly discuss some issues that were not covered in this paper and will be the focus of further research.

4.1. Transformation issues

The initial version of the proposed database design environment discussed in this paper will implement only a small number of transformations, in order to explore the following issues before moving to a more thorough implementation:

- Information "gain" during a transformation must be dealt with in some way. Some of this information may be able to be generated automatically; that which cannot will require interaction with the user.
- Technique transformations are likely to be more difficult to deal with than scheme transformations, because of the greater likelihood of differences in the information that can be represented.
- If some schemes do not fully adhere to the technique they express, this raises the question of how useful the transformations to, from and between these schemes are.
- When a representation is modified, are the changes automatically propagated to other representations, or should the user be required to do this manually? Another related issue is revision management.

4.2. Formal analysis of transformation quality

Analysis of transformation quality has to this point been relatively informal and *ad hoc*. This will be remedied in the near future by performing a more formal analysis using the *information capacity* (Miller, Ioannidis et al. 1993) of various representations as a qualitative measure of representation equivalence. *Schema intension graphs* (Miller, Ioannidis et al. 1994) will be used to model schema instances from each representation, and from this determine the equivalence (or lack thereof) of the representations. The results of this analysis should be available by the time of publication.

4.3. Additional representations

This paper has only discussed three main representations: the entity-relationship approach, the relational model and functional dependencies. There are many other formal and semi-formal representations that could be included in such an environment, such as process modelling, object modelling, formal specifications and so on. Inclusion of informal representations is another line of research, although this could be difficult because of the unstructured nature of many such representations.

4.4. Schema generation

The issue of schema generation from multiple representations was introduced in Figure 6 but not explored further. If we discard the notion of a uniform representation, generating a schema becomes a multi-step process, in which we must decide how to use the different representations. One approach is to refine the generated schema in a stepwise fashion (Figure 7) using the information contained within each representation description. We will denote the refinement operator by the symbol $\mapsto$.

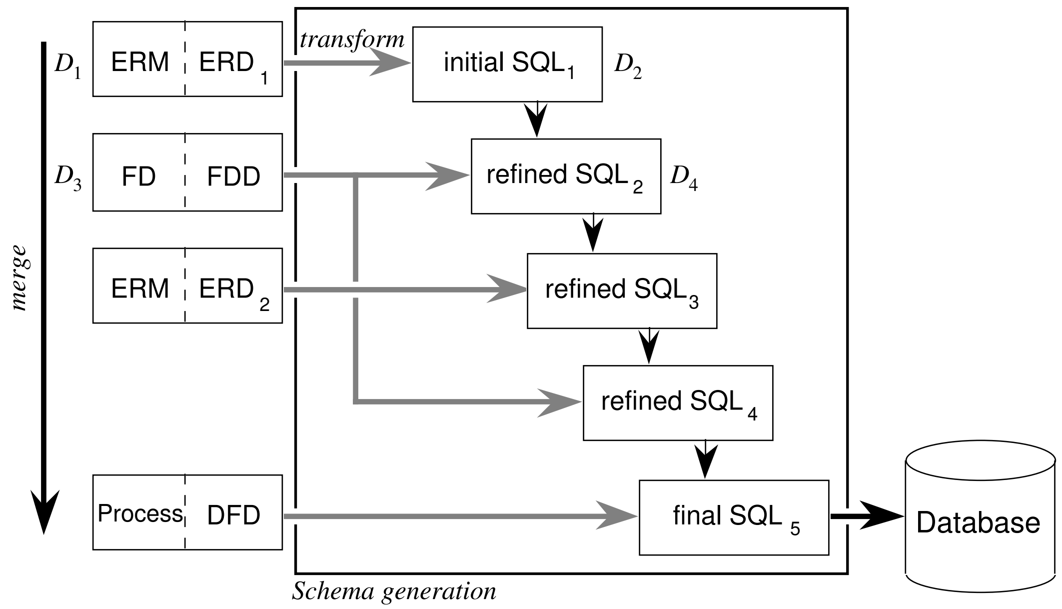

Figure 7. Schema generation from multiple representations.

The box labelled "schema generation" in Figure 7 corresponds to the "black box" of Figure 6. The boxes labelled D_1 through D_4 in Figure 7 correspond to the descriptions used in the first step of the refinement process. In this example, the "source" representations consist of two ER representations, a functional dependency representation and a process model representation expressed using a data flow diagram. The "target" representation for schema generation is SQL. The first step is to generate an initial SQL schema from the first ER representation (D_1). The next step is to normalise this description using the information contained within the FD representation (D_3). We denote these transformations as:

$$\left.\begin{array}{r} D_1(V, ERM, ERD_1) \rightarrow D_2(V, RM, SQL_1) \\ D_3(V, FD, FDD) \end{array}\right| \rightarrow D_4(V, RM, SQL_2)$$

This refinement process continues for each representation used to describe the viewpoint, as shown in Figure 7. Because of the extra information that multiple representations provide, this could result in a "better" schema than if we had generated it from just a single representation, although it must be remembered that not all of this extra information may be able to be represented in the target schema.

There is no particular significance to the order of refinements used in the example above; indeed, a different order might prove more appropriate or efficient. In addition, there is no need for all the steps to output the target representation. Automatic optimisation of this process could be a particularly interesting area of research. It is also possible that changing the refinement order may result in a different final schema. Again, this may indicate a problem with the integrity of the design.

5. Summary

In this paper, we discussed the concept of describing a developer viewpoint using multiple representations. The advantage of such an approach is that the viewpoint may be described more fully by using multiple representations than by using a single representation. Such an approach could be implemented in a database design environment by enabling the environment to support the transformation of viewpoint descriptions from one representation to another. We discussed the issues raised by such transformations, such as the integrity of the design produced and the quality of the transformations in terms of information gained and/or lost, and introduced a notation for describing operations on viewpoint descriptions. A possible architecture for a database design environment that uses multiple viewpoint representations was proposed and possible directions for future research were discussed.

6. References

Brooks, F. P. (1975). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, Massachusetts.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, **13**(6).

Cunningham, R. J., A. C. W. Finkelstein, S. Goldsack, T. S. E. Maibaum and C. Potts (1985). Formal requirements specification — The FOREST project. In *Proceedings of the Third IEEE International Workshop on Software Specification and Design*, London.

Darke, P. and G. Shanks (1995). Viewpoint development for requirements definition: Towards a conceptual framework. In *Proceedings of the 6th Australasian Conference on Information Systems (ACIS '95)*, Perth, Australia, pages 277-288.

Date, C. J. and H. Darwen (1993). *A Guide to the SQL Standard*. Addison-Wesley, Reading, Massachusetts, third edition.

Easterbrook, S. M. (1991a). *Elicitation of requirements from multiple perspectives*. PhD thesis, Imperial College of Science Technology and Medicine, University of London, London.
[http://research.ivv.nasa.gov/~steve/papers/thesis/thesis.ps]

Easterbrook, S. M. (1991b). Handling conflict between domain descriptions with computer supported negotiation. *Knowledge Acquisition: An International Journal*, **3**(4): 255-289.

Easterbrook, S. M., A. C. W. Finkelstein, J. Kramer and B. A. Nuseibeh (1994). Co-ordinating distributed ViewPoints: the anatomy of a consistency check. *Journal of Concurrent Engineering: Research and Applications*, **2**(3).

Easterbrook, S. M. and B. A. Nuseibeh (1995). Managing inconsistencies in an evolving specification. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE'95)*, York, UK, pages 48-55.

Easterbrook, S. M. and B. A. Nuseibeh (1996). Using ViewPoints for inconsistency management. *Software Engineering Journal*, **11**(1): 31-43.

Evergreen Software Tools (1995). *EasyCASE® Methodology Guide*, Evergreen Software Tools, Inc., Redmond, Washington, version 4.2.

Finkelstein, A. and I. Sommerville (1996). The viewpoints FAQ. *Software Engineering Journal*, **11**(1): 2-4.

Finkelstein, A. C. W., M. Goedicke, J. Kramer and C. Niskier (1989). ViewPoint oriented software development: Methods and viewpoints in requirements engineering. In *Proceedings of the Second Meteor Workshop on Methods for Formal Specification*, Springer-Verlag.

Klein, H. K. and R. A. Hirschheim (1987). A comparative framework of data modelling paradigms and approaches. *The Computer Journal*, **30**(1): 8-15.

Kotonya, G. and I. Sommerville (1996). Requirements engineering with viewpoints. *Software Engineering Journal*, **11**(1): 5-18.

Leite, J. C. S. P. and P. A. Freeman (1991). Requirements validation through viewpoint resolution. *IEEE Transactions on Software Engineering*, **17**(12): 1253-1269.

Miller, R. J., Y. E. Ioannidis and R. Ramakrishnan (1993). The use of information capacity in schema integration and translation. In *Proceedings of the Nineteenth International Conference on Very Large Data Bases (VLDB)*, Dublin, Ireland, pages 120-133.

Miller, R. J., Y. E. Ioannidis and R. Ramakrishnan (1994). *Schema intension graphs: A formal model for the study of schema equivalence*. Technical report CS-TR-94-1185, Department of Computer Sciences, University of Wisconsin. [http://www.cs.wisc.edu:80/Dienst/Repository/2.0/Body/ncstrl.uwmadison%2fCS-TR-94-1185/postscript]

Pascoe, R. T. and J. T. Penny (1990). Construction of interfaces for the exchange of geographic data. *International Journal of Geographical Information Systems*, **4**(2): 147-156.

Sallis, P., G. Tate and S. MacDonell (1995). *Software Engineering: Practice Management, Improvement*. Addison-Wesley, Reading, Massachusetts.

Shaw, M. L. G. and B. R. Gaines (1989). Knowledge acquisition: Some foundation, manual methods and future trends. In *Proceedings of the Third European Workshop on Knowledge Acquisition for Knowledge-Based Systems (EKAW-89)*, Paris.

Smith, H. C. (1985). Database design: composing fully normalized tables from a rigorous dependency diagram. *Communications of the ACM*, **28**(8): 826-838.