Database Interoperability Through State Based Logical Data Independence

Michael Johnson
Department of Computing
Macquarie University
Sydney, Australia
Email: mike@ics.mq.edu.au

Robert Rosebrugh
Department of Computer Science
Mount Allison University
NB, Canada
Email: rrosebru@mta.ca

Abstract

Computer supported cooperative work (CSCW) involving business-to-business transactions depends more and more upon database interoperability. The design of inter-business CSCW when the businesses are already operating independent systems depends either upon effective reverse engineering (to properly discover the semantics underlying each organisation's systems and through that to develop appropriate matches for interbusiness software), or upon sufficiently rich semantic models and good database management system support for logical data independence (to allow database updating through a logical view). This paper takes the second approach presenting a rich semantic data model that the authors have been developing and have used successfully in a number of major consultancies, and a new approach to logical data independence and view updatability based on that model. We show how these approaches support database interoperability for business-to-business transactions, and, for CSCW within an organisation, how they support federated databases.

Keywords: Logical data independence, category theory, semantic data modelling

1 Introduction

The design of information systems has long been recognised to be a delicate and difficult task, and it has utilised a plethora of methodologies ranging from traditional relational database design [5] and entity relationship (ER) modelling [4], through functional data models [25], knowledge based systems [3] and UML [22]. This paper addresses the need to design *interactions* between information systems both for business to business transactions, and for computer supported collaborative work within individual organisations.

Recently Myopoulos has argued [19] that interactions can best be designed when rich semantic models are available for each of the information systems. But Myopoulos has also lamented the paucity of successful semantic data modelling paradigms. A successful semantic data modelling paradigm needs to be simple, clean, and mathematically well-founded (like the relational data model). It needs to have the power to make semantically rich specifications and to model a wide range of real world constraints. And it should have an easy to understand graphical representation.

Category theory ([2], [18], [28]) is a branch of mathematics renowned for its semantic power, its simple axiom set, and its use of graphical techniques. It has been widely used for specification in computer science in, for example, abstract data types [14], [15], semantics of programming languages [23], [24], and functional programming [1].

The authors and their coworkers have, over a number of years, been developing a semantic data modelling paradigm based on category theory. The categorical specification of information systems (see for example [16], [17]) has been motivated by categorical universal algebra (a categorical specification technique used in mathematics) and has been successfully utilised in a number of major consultancies [8], [7], [26]. In fact the technique grew out of the needs of a very large information system specification consultancy [6]. Other related work includes [20], [21], [12], [13].

Categorical information system specification satisfies the criteria to be a semantic data modelling paradigm that would well support information system interactions. This paper investigates that possibility and explains the requisite theory of categorical logical data independence.

We argue that information system interaction is best supported by a strong notion of logical data independence, so our first task, after reviewing categorical information system specification in Section 2 is to develop a treatment of logical data independence in our category theoretic framework. This is done in Section 3. In Section 4 we explore the transmission of updates across the data independence boundary, and in Section 5 we show how the theory developed in the earlier sections can be used to link interacting or federated information systems.

2 Category Theoretic Information System Specification

This section briefly reviews category theoretic information system specification. More details can be found in [16]. Definitions and elementary properties of commuting diagrams, limits, and coproducts, can be found in most introductions to category theory, including [2] and [28]. We will assume familiarity with ER terminology in this section [27].

In outline, an information system is specified in the category theoretic data model by giving a *schema*, that is a graph, roughly corresponding to an ER graph, and a set of (categorical) constraints. The constraints take three forms:

1. Commuting diagrams are pairs of paths in the graph with common origin and destination.

2. Limit constraints specify that a certain node in the graph is to act as the "limit" of a specified diagram in the graph.

3. Coproduct constraints specify that a certain node in the graph is to act as the "coproduct" of specified nodes in the graph.

A database, sometimes called a database state or instance, is an assignment of, for every node in the schema a finite set, (the set of instances or values of that entity or attribute), and for every arrow in the schema a function between the corresponding sets, (the relationships among the entity instances, or the attribute values corresponding to the instances), such that

1. The commuting diagrams do indeed commute as diagrams of corresponding functions

2. The sets assigned to limit nodes are indeed the limits of the corresponding specified diagrams of functions

3. The sets assigned to coproduct nodes are indeed coproducts (disjoint unions) of the sets assigned to the corresponding specified nodes.

In other words, a database state for a categorical data model schema is a diagram of sets and functions which is the same shape as the graph, and which satisfies the constraints. When the context is clear, we will refer to a database state E for a category theoretic data model schema $\mathbb{E}$ as simply a state E of $\mathbb{E}$.

Figure 1 is an example schema. The graph is a fragment of an e-business schema. The full schema includes among its constraints the requirements that the two triangles commute, that the square be a pullback (a kind of limit), and that the Product node be the coproduct of Physical product and Virtual product (electronically available software, music, etc). The constraints ensure that: Deliveries only take place to fill specific Orders (the commuting triangles); All Products, and only those Products, which have a Physical location are Physical products (the pullback); All Products are either Physical products or Virtual products (the coproduct). The Location entity stores both warehouse locations and URLs. Also shown is a subtype relation: Physical location is a subtype of Location, and the function between them is guaranteed to be injective by another limit specification. It follows from general properties of pullbacks and coproducts that the functions corresponding to the other isa arrows are also injective.

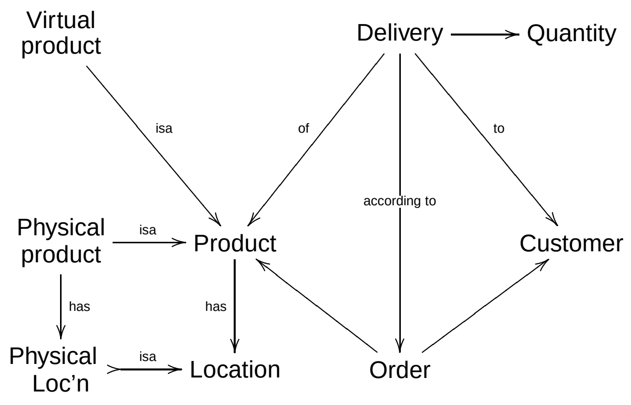

Figure 1. A fragment of an e-commerce category theoretic data model schema

Schemata can be interrelated using schema maps. A *schema map* is a graph morphism between the corresponding graphs which maps each of the constraints on the first schema graph to a constraint (already) specified in the second schema graph.

Incidentally, a category has an underlying schema. Its graph is the underlying graph of the category, and its constraints are all of the constraints that happen to be true in the category: all of the commuting diagrams, all of the limits, and all of the coproducts. When we refer to a schema map into a category (as we will in Section 3) we in fact mean a schema map into the underlying schema of the category.

A schema generates a *classifying category*. Roughly speaking it is the smallest category containing the schema, satisfying the constraints, and closed under finite limits and finite coproducts. The classifying category has important technical uses, and, as we are about to see, is important for logical data independence too.

3 Logical Data Independence

Physical data independence allows a user to work with a database without needing to concern themselves with how the data are physically stored in the database. *Logical data independence* allows a user to work with a database without needing to concern themselves with how the data are logically arranged in the database — the user is insulated from the schematic structure of the database. With logical data independence the user can view and manipulate data in an arrangement, a logical data structure, which is independent of the actual logical structure of the database.

Logical data independence is important because it allows us to modify a database schema, perhaps adding more attributes, or extra entities and relations as a business evolves, but to continue to use the same applications programs, and to present staff who do not need to deal with the new data with the same interface that they were using before the change.

Many systems include view mechanisms that provide a degree of logical data independence. Unfortunately view mechanisms sometimes suffer restrictions, both in what views can be defined and in how the viewed data can be manipulated.

True logical data independence should have the following properties:

1. A logically independent view should be able to contain any data that can be derived from the data stored in the underlying database. We call this *the universality principle*. Of course particular views are sometimes constructed to *limit* access to data. A view can only see the data that it is designed to include. The point of the principle is that in designing a view we should be able to choose to include in that view any data from the universe of data available from the database.

2. A logically independent view should be able to be structured, queried and manipulated as if it were an independent database. We call this *the modularity principle*. The point of the principle is that as users we should not be able to distinguish a logically independent view from a database, and so we should for example be able to define logically independent views of our view, etc, and as designers we should be able to structure the data in the view in any manner (compatible with the underlying data) as if we were designing an independent database.

Notice that detailed interpretation of the two principles takes place relative to the data model employed. The universe of data available from a database depends upon the queries that are available, that is, depends upon the queries that are supported by the data model in use. Similarly the modularity principle assures us that we can structure viewed data as if it were a database, and the range of structures we have available for our databases depends upon the data model we are using.

We now present our approach to logical data independence in the category theoretic data model.

Definition 1 Let $\mathbb{E}$ be a category theoretic schema and let $Q(\mathbb{E})$ be its classifying category. A *view* of $\mathbb{E}$ is a schema $\mathbb{K}$ and a schema morphism $k : \mathbb{K} \longrightarrow Q(\mathbb{E})$. For a given view, we will refer to the database corresponding to the schema $\mathbb{E}$ as the *underlying database* of the view.

How is this a view of a database? Well, $\mathbb{K}$ is a schema, so it tells us how the view is structured as a (virtual) database, and the schema map k indicates for each node in $\mathbb{K}$ where to find the corresponding data in the underlying database.

Do views support true logical data independence? Yes. We consider each principle in turn.

Universality: It was noted in [9] that $Q(\mathbb{E})$ has objects (nodes) corresponding to all the queries that can be made on a database with schema $\mathbb{E}$. So in designing a view we can build nodes whose value under k, whose data, is the result of any query on the underlying database.

Modularity: In the category theoretic data model a database is presented by giving a schema. Thus $\mathbb{K}$, the schema for the view, can be structured, queried, or manipulated, as if it were an ordinary database. (But of course the data available through $\mathbb{K}$ will, since it has come from the underlying database, always be subject to the constraints in $\mathbb{E}$. The implications of this will be taken up in the next section.)

Finally, a word of caution about k. A schema map carries constraints to constraints. Thus, $\mathbb{K}$ cannot include constraints that have no corresponding constraints in $\mathbb{E}$. This is exactly as we would expect — it shouldn't be possible to require constraints in the view which are not required in the underlying database since the data from the underlying database, which might violate those constraints, is just inherited by the view. This doesn't violate modularity. In fact, it clarifies the meaning of "compatible with the underlying data" in the statement of the modularity principle.

It should also be noted that the constraints just mentioned have nothing to do with the queries used to define the view. A view of the orders from company X does not include a constraint that says orders come only from company X. Instead it takes its values from queries of the form

```
select  ...  where COMPANY = X.
```

4 The Transmission of Updates

Now we begin to consider in earnest issues of database interoperability. If K is a logically independent view of a database D, we should be able to insert and delete in K. But in reality K is a view of some of the data derivable from D, so an insert or delete in K must transmit some change to the underlying database D. This change of the data stored in D must be arranged so that the change of the *view* of the data corresponds to the intended insert or delete.

Unfortunately, not all view inserts and deletes are transmissible in this way.

Let's consider a (simple) example or two. Suppose $\mathbb{K}$ and $\mathbb{E}$ are the same, except that $\mathbb{E}$ includes an extra constraint that is not required in $\mathbb{K}$ (remember that each constraint in $\mathbb{K}$ is mapped by k to a constraint in $\mathbb{E}$, but nothing prohibits extra constraints in $\mathbb{E}$). Now an insert in the view which violates the extra constraint is acceptable to the (virtual) view database, but cannot be transmitted to the underlying database (because it violates the extra constraint).

For another example consider the schema

$$\begin{tikzcd}\mathsf{Age} & \mathsf{Person} \arrow[l] \arrow[r] & \mathsf{Address}\end{tikzcd}$$

with no constraints. Let $\mathbb{K}$ be $\mathsf{Person} \longrightarrow \mathsf{Address}$ (with k the evident inclusion). An insert into Person in $\mathbb{K}$ requires the specification of the person's address. But upon transmission to D we find that we cannot make such an insertion, because we need to specify the person's age too.

Do these violate the modularity principle? Arguably they do. But it has also been suggested that the views are still independent databases, it's just that they contain extra hidden constraints including in the first case the "extra" constraint, and in the second case a constraint prohibiting inserts into Person.

If we take this point of view, we need to characterise the hidden constraints. What inserts and deletes are transmissible from views to underlying databases? This is known as the *view update problem* [11, Chapter 8].

There has been considerable confusion over the view update problem because people have sought to answer it in terms of schemata. We argue that it is really a property of the current *state* of a database, and define transmissible as follows.

Definition 2 Suppose we have a view $k : \mathbb{K} \longrightarrow Q(\mathbb{E})$ and a current state D for the database corresponding to $\mathbb{E}$. A view insert is *transmissible* if there is a unique minimal insert in D (the underlying database) which achieves the view insert.

This can be cast into mathematical terms, and then general transmissibility results for particular schemata can be proven if desired [10], but that is not the immediate purpose of this paper. Instead we move to how we can use state-based transmissibility and logical data independence to support database interoperability and CSCW.

But first, a warning about "unique minimal insert" in the definition. The phrase is easily misinterpreted. It in fact means that among all of the inserts $D \rightarrowtail D''$ which achieve the view insert, there is one $D \rightarrowtail D'$ which is initial in the full subcategory of the slice category under D ([2]).

5 Constructing Systems

Suppose we have two or more databases that need to interoperate. The databases might be used in business-to-business web-based transactions, or just in collaborative work between two divisions inside one organisation. Throughout this section we will assume that the databases have been based on the category theoretic data model. Thus we have a rich description of the data via schemata which include constraints.

We seek to design systems of interoperation.

Suppose two databases which need to interoperate have schemata $\mathbb{E}$ and $\mathbb{F}$, and current states E and F respectively.

The first problem in supporting interoperability is that common data stored in the two databases might have different names. Typically designers scan data dictionaries, or quiz human representatives of each of $\mathbb{E}$ and $\mathbb{F}$, to try to discover matches. Our methodology has little to add to this process. Looking at the structural arrangement of nodes in each graph, and in constraints, can suggest candidates for matching, but the added value is small.

A more serious problem usually arises next. Although the databases do indeed deal with common data, and should be well positioned to interoperate, there are too few matches found. This arises because the data can be stored in structurally different ways in each of the databases. For example, one database might have separate nodes for each of several different types of product, while the other has a single node for all products. Or one database might only store the products which are at a particular location, while the other stores in one node the products from all locations and has an attribute to record their locations. This is where we use logical data independence.

Recall that a view consists of $k : \mathbb{K} \longrightarrow Q(\mathbb{E})$, and that $Q(\mathbb{E})$ contains all queries that can be performed on $\mathbb{E}$. So we can construct views that see the coproduct of all of the different types of product, or the subtype of products at all locations obtained by pulling back over the particular location, that is, the result of a query of the form

```
select  ...  where LOCATION = ...
```

In this manner we construct a logically independent structure, a view, which contains matchable nodes from each of $Q(\mathbb{E})$ and $Q(\mathbb{F})$, and schema maps

$$\begin{tikzcd}Q(\mathbb{E}) & \mathbb{K} \arrow[l] \arrow[r] & Q(\mathbb{F}).\end{tikzcd}$$

Next we explore state-based transmissibility across this structure (called a *span*) of schema maps. Consider arbitrary legal states E and F of $\mathbb{E}$ and $\mathbb{F}$. An insert or delete in E affects directly the state K of $\mathbb{K}$ under the view k. We explore the transmissibility of this change of the state of K to the state F. Frequently we can prove lemmas that assure us that all such changes are transmissible. In that case we have achieved interoperability, and the schema maps, expressed in SQL terms, together with the construction used in the proof of the lemmas, provides the "code" for the interoperability mechanism.

In some cases the failure to prove transmissibility identifies specific cases where interoperability fails, and these can be the subject of negotiations between the businesses involved to determine business rules or database changes that will assure interoperability.

Another interesting use of this methodology arises when an organisation seeks to construct an enterprise model, and has individual data models in divisions. The interoperability approach can be used to construct a diagram, frequently of the form

$$\begin{tikzcd} \mathbb{E} & \mathbb{K} \arrow[l] \arrow[r] & \mathbb{E}' & \mathbb{K}' \arrow[l] \arrow[r] & \mathbb{E}'' \ldots \end{tikzcd}$$

The *colimit* of this diagram forms the enterprise data model and is an example of a federated database.

6 Conclusion

This paper has developed an approach to CSCW database support which achieves database interoperability and database federation through designated views. The views depend on a new formulation of logical data independence, which, unusually, takes a state-based approach to the transmissibility of updates.

The methodology is founded on a category theoretic data model which has already been well-tested in industrial consultancies. This foundation provides relatively rich semantics which aid in properly designing interoperable views, and a well-understood mathematical basis which supports both constraint definition and the proofs required to guarantee the transmissibility of updates.

7 Acknowledgements

The research reported here has been supported in part by the Australian Research Council, the Canadian NSERC, and the Oxford Computing Laboratory.

References

[1] A. Asperti and G. Longon. *Categories, Types and Structures: An introduction to category theory for the working computer scientist*. MIT Press, 1991.

[2] M. Barr and C. Wells. *Category theory for computing science*. Prentice-Hall, second edition, 1995.

[3] M. L. Brodie and J. Myopoulos. *On Knowledge Base Management Systems*. Springer-Verag, 1986.

[4] P. P. -S. Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, 2:9–36, 1976.

[5] E. F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13:377–387, 1970.

[6] C. N. G. Dampney and Michael Johnson. TIME Compliant Corporate Data Model Validation. Consultants' report to Telecom Australia, 1991.

[7] C. N. G. Dampney and Michael Johnson. Fibrations and the DoH Data Model. Consultants' report to NSW Department of Health, 1999.

[8] C. N. G. Dampney, Michael Johnson and G. M. McGrath. Audit and Enhancement of the Caltex Information Strategy Planning (CISP) Project. Consultants' report to Caltex Oil Australia, 1993.

[9] C. N. G. Dampney, Michael Johnson, and G. P. Monro. An illustrated mathematical foundation for ERA. In *The unified computation laboratory*, pages 77–84, Oxford University Press, 1992.

[10] C. N. G. Dampney, Michael Johnson, and Robert Rosebrugh. View Updates in a Semantic Data Model Paradigm. To appear, ADC2001, IEEE Press, 2001.

[11] C. J. Date. *Introduction to Database Systems*. Addison-Wesley, fourth edition, 1986.

[12] Zinovy Diskin and Boris Cadish. Algebraic graph-based approach to management of multidatabase systems. In *Proceedings of The Second International Workshop on Next Generation Information Technologies and Systems (NGITS '95)*, 1995.

[13] Zinovy Diskin and Boris Cadish. Variable set semantics for generalised sketches: Why ER is more object oriented than OO. In *Data and Knowledge Engineering*, to appear, 2000.

[14] J. Goguen, J. W. Thatcher and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, *Current Trends in Programming Methodology IV*, 80–149, Prentice-Hall, 1978.

[15] J. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *JACM*, 24:68–95, 1977.

[16] Michael Johnson and C. N. G. Dampney. On the value of commutative diagrams in information modelling. In The Unified Computation Laboratory, eds Rattray and Clarke, *Springer Workshops in Computing*, Springer-Verlag, 1994.

[17] Michael Johnson, Robert Rosebrugh, and R. J. Wood. Entity-relationship models and sketches. Submitted to *Theory and Applications of Categories*, 2000.

[18] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics 5, Springer Verlag, 1971.

[19] John Myopoulos. Next generation database systems won't work without semantics! Panel session, *SIGMOD Record*, 27:497, 1998.

[20] F. Piessens and Eric Steegmans. Categorical data specifications. *Theory and Applications of Categories*, 1:156–173, 1995.

[21] F. Piessens and Eric Steegmans. Selective Attribute Elimination for Categorical Data Specifications. Proceedings of the 6th International AMAST. Ed. Michael Johnson. *Lecture Notes Computer Science*, 1349:424-436, 1997.

[22] J. Rumbaugh, I. Jacobson and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, 1999.

[23] D. Scott. Continuous lattices. *Lecture notes in Mathematics*, 274:97–136, 1972.

[24] D. Scott. Domains for denotational semantics. *Lecture notes in Computer Science*, 140:577–613, 1982.

[25] D. Shipman. The functional data model and the data language DAPLEX. *ACM TODS*, 6:140–173 1981.

[26] G. Southon, C. Sauer, and C. N. G. Dampney. Lessons from a failed information systems initiative: issues for complex organisations. *International Journal of Medical Informatics*, Elsevier Science, 1999.

[27] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*. Volume 1, Computer Science Press, 1988.

[28] R. F. C. Walters. *Categories and Computer Science*. Cambridge University Press, 1991.