On the value of commutative diagrams in information modelling

Michael Johnson and C.N.G. Dampney
School of Mathematics and Computing, Macquarie University
Sydney, AUSTRALIA

Abstract

Category theory has been widely used in computer science, but usually in a very sophisticated manner. This paper argues that elementary category theoretic notions can have important value in the "real world" of software engineering. Perhaps the most elementary categorical notion is that of commutative diagram. Drawing on experience from several applications of category theory to information modelling in major business enterprises we show how commutative diagrams have been used to develop new methodologies in ER-modelling, constraint specification and process modelling. They also suggest new but as yet untested techniques for information model partitioning and information system architecture. The methodologies described here have a firm theoretical basis using the recently isolated theory of lextensive categories and this basis is briefly outlined.

1 Introduction

Category theory is an algebraic methodology that has had extensive use in computer science. It has been particularly valuable in distinguishing syntax and semantics by the use of classifying categories and structure preserving functors into the category of sets (in a manner analogous to its use in universal algebra). This paper outlines a classifying category setting for information models based on entity-relationship (ER) diagrams and traces several of its consequences for information modelling methodology.

There have been several category theoretic treatments of information systems (see [5] [6] [8] [9]) but to the authors' knowledge none of them has influenced information modelling methodologies until now. The results reported here are predominantly empirical, and are based on consultancy work that we have undertaken for Telecom Australia, Caltex Oil Australia and several smaller enterprises.

The most remarkable feature of the work is that one of the simplest of categorical notions — commutative diagrams — played a central role in the discovery of the new methodologies.

The paper is organised as follows. Section 2 reviews the dominant information modelling methodology which is entity-relationship modelling. In Section 3 we review the definition of a category and indicate how an ER-model is essentially a category—the *classifying category* or *theory* for the information model. A brief analysis of this view shows that our classifying categories are at least boolean and lextensive [2] and that the category theoretic treatment yields immediately a query language. This is the main theoretical section of the paper. In Section 4 we note that the main difference between an ER-model and its classifying category is the specification of commutative diagrams which amount to integrity constraints upon data which can be stored in the information model. This has led to a change in the main methodologies by giving constraint specification a much greater role in the development of information models. We show how to treat both static constraints and dynamic constraints and we illustrate these using examples drawn from industry. Section 5 treats process modelling which is traditionally the next stage in the development of a system after information (ER) modelling and it shows how the categorical treatment greatly simplifies process modelling. Finally Section 6 records implications of the category theoretic framework for developing different user views of an information model and for the underlying architecture of information systems.

Overall our approach has become known as the Federated Information System (FIS) approach to information modelling and information system engineering.

Acknowledgments: The authors gratefully acknowledge the Australian Research Council (ARC) and Caltex Oil Australia for supporting this research.

2 Information Models

Information models are widely used in industry because they are usually the first step in the development of an information system, and because they can provide a corporate overview and are hence valuable in strategic planning. An information model is a formal description of those aspects of reality about which a business needs to maintain information. Thus information modelling is about abstracting from reality with a particular formalism in mind, and the choice of formalism can affect the resulting model.

There are several formalisms that are used as a basis for information models but by far the dominant technique is called *Entity-Relationship* (ER) *modelling* [3].

The ER approach is a graphical modelling technique. An *entity* is a class of something about which the business needs to store information. Examples might include CUSTOMER, EMPLOYEE, ORDER, INVOICE and PRODUCT. Each entity will correspond to a set of things at a particular point in time (for example the current set of employees). The information that we store about entities comes in two forms: there are *relationships* between entities (for example an order may be for several products and a product may appear on several orders so there is a many-to-many relationship between PRODUCT and ORDER) and entities have certain *attributes* (for example a product may have a product number and a price, an employee has a name, an address, a salary and so on). Often one attribute for each entity is treated as the *key attribute* so that for example a product may be always accessed via its product number. Entities are usually represented graphically as rectangular boxes, relations as lines joining the boxes (with "crows-feet" to indicate possibly multi-valued relations when necessary), and attributes as oval boxes. An example is shown in Figure 1.

A more detailed example appears in Figure 2 which is a strategic model of the Australian coal industry.

Figure 1: A simple ER model fragment

The graphical nature of ER-models is a very important aspect of their popularity. Other specification techniques such as Z are more powerful but harder to learn. The great value of a graphical model is that an analyst can show it to businesspeople and with only a brief explanation they can understand and if necessary correct the model.

There is an extensive methodology of ER-modelling including the reduction of models to various normal forms. The details need not concern us here except for one aspect: Many-to-many relationships can always be transformed into two many-to-one relationships by the introduction of a new entity. For example in the model above we can introduce an entity ORDER_LINE. One instance of this entity will be the order of a particular product on a particular order. Thus there will be many-to-one relations (functions) between ORDER_LINE and PRODUCT and between ORDER_LINE and ORDER. (This is just the usual “tabulation” of a relation.)

3 Category theory

A category consists of a collection of *objects* and a collection of *arrows*, with each arrow having a specified source and target among the objects (this much is just a directed graph) together with a *composition* of arrows, defined whenever the arrows have a common source and target, which is associative and has identities.

Thus a category may be thought of as a directed graph together with information about composition. This information may be expressed by giving a set of relations (eg f composed with g is equal to h composed with k) and those relations are often expressed as *commutative diagrams* (a diagram is said to commute if any two composable paths of arrows in the diagram with common start point and common end point have equal composites).

Examples of categories include the category **set** whose objects are sets and whose arrows are functions between sets; **grp** whose objects are groups and whose arrows are group homomorphisms; more generally **T-alg** whose objects are algebras from some theory T and whose arrows are T-homomorphisms; and numerous *small categories* which can be generated by drawing a directed graph, adding identities at each vertex, and specifying composites for composable pairs of arrows.

One of the great advantages of category theory is that it has provided a graphical framework for much of mathematics. Many properties that seem to be about the internal structure of objects such as being a one element set or being the cartesian product of two sets, can be characterised by universal properties of arrows and these permit graphical arguments to prove theorems.

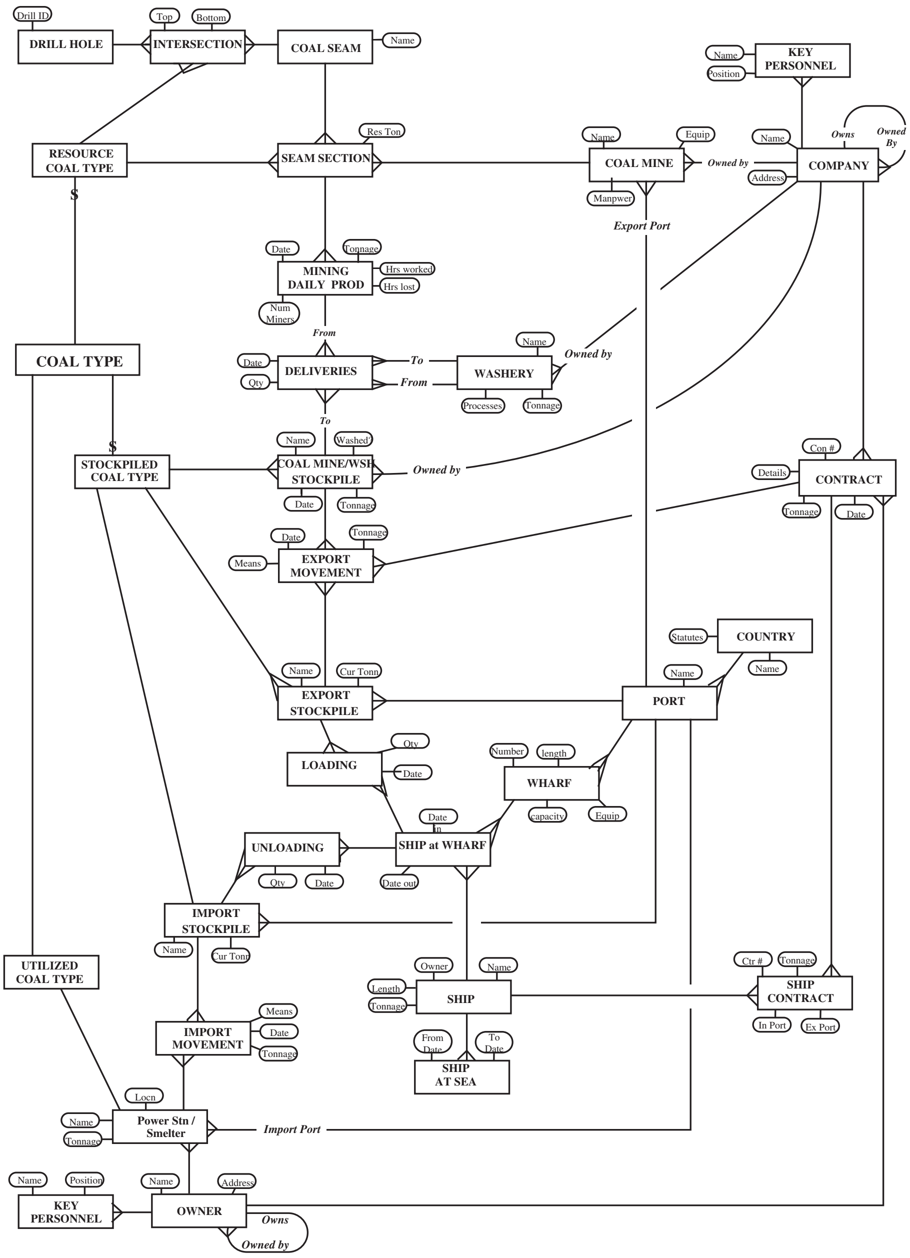

Figure 2: A strategic model of the Australian coal industry

Further examples and definitions of specific universal properties such as pullback, terminal object and coproduct can be found in any of the basic texts [7] [10] [1].

3.1 A category theoretic view of an ER-model

We aim now to show how an ER-model is essentially a category. This is motivated by the categorical treatment of universal algebra above (the T-**alg** example) and is treated in full in [5]. By analogy with universal algebra we will call the category the *theory* or *classifying category* of the ER-model.

Consider an ER-model, normalised as described at the end of the preceding section so that all relations are many-to-one. This model may be viewed as a directed graph whose vertices are the entities and attributes of the model and whose arrows are the relationships oriented from the many-valued entity to the one-valued entity together with arrows from each entity to each of its attributes. Notice that if the vertices of this graph are thought of as sets, and the arrows as functions, then the intended semantics of the ER-model is still well represented here (and this can be made formal via a functor to **set** in the usual way).

It remains to consider composition. Since the many-to-one relations in the model are intended to represent real world many-to-one relations (functions) there are real world compositions and we argue that these should be represented in the model. Many of the compositions are free in the sense that formal composites can just be added to the model (or indeed left out since such formality can be added later), but when there is a closed loop of arrows it is important to determine, by considering the real world semantics, whether the diagram commutes. Once this has been done for all possible composites we have constructed a classifying category for the ER-model.

It is remarkable that extant ER-modelling methodologies have ignored this question of commuting diagrams. Typically an analyst spends a great deal of time and effort developing a model and eventually passes it to a programmer to implement. Often it is important that the resultant program check the constraint implied by the commutativity of certain diagrams, but since the analyst has not recorded which diagrams commute it is up to the less experienced programmer to try to reconstruct the intended semantics and to decide whether a given diagram should commute!

In fact, in our experience, searching for commutative diagrams actually results in a better ER-model because it often clarifies the nature of relationships and because it provides a test of the model as it is being developed. A detailed example based on Figure 2 appears in Section 4.1.

The search for commutative diagrams has also found its way in disguise into some special rules in standard methodologies. The “double-vee” rule used in the British Standards methodology is one instance of a rule which helps to locate commutative diagrams (although it is not expressed that way).

In summary, we view the specification of which diagrams commute in an ER-model as an important part of the information modelling methodology. We are developing CASE tools which will provide assistance with this process.

3.2 Classifying categories and lextensive categories

Universal algebra suggests a better version of the classifying category discussed above. Often an algebraic theory can be presented in several different ways, but there is a single canonical classifying category (up to equivalence of categories) obtained by taking any one of the presentations and "closing it up" under certain basic operations like taking limits. Similarly we would expect the classifying category of an information system to satisfy certain basic exactness properties and the category described in Section 3.1 is just a presentation for the canonical classifying category.

So, what basic exactness properties are required? We need a terminal object I, and arrows $I \to A$ will be used to specify instances of the entity A. We need finite coproducts for two reasons. First, entities often have substructure which is best indicated by coproducts (so for example in a small retail business the entity EMPLOYEE might be the coproduct of the entities DRIVER, SALESPERSON, CLERICAL_STAFF and MANAGEMENT). Secondly, attributes are fixed sets (so for example PRODUCT_NO might be the set of all four digit numbers—of course most of these numbers won't be used at any particular point in time, but the relationship between PRODUCT and PRODUCT_NO allows us to see which ones are currently valid product numbers). Thus attributes are usually $\sum^n I$ for some n ($n = 10000$ in our product numbers example). This is technically very important since the injection $i_k : I \to \sum^n I$ allows us to pick out attribute number k from which, if the attribute is a key attribute, we can obtain information about a particular instance of the corresponding entity. Finally we need pullbacks, both to allow us to compose relations and to allow us to access the entity instances with particular attribute values.

Furthermore we expect the coproducts to behave well. They should be disjoint and universal. Thus in the presence of pullbacks and a terminal object we expect our classifying category to be a *lextensive category* [2].

Finally, we expect subobjects to be complemented with $I + I$ acting as a subobject classifier. An extensive category with this property is called a Boolean category and is necessarily lextensive.

3.3 The query language

For use in Section 4 it is worth noting that the internal logic of the lextensive classifying category of an information model forms a query language for that model. Thus the standard queries arise as objects of the classifying category.

The semantics of the information model will be given by lextensive functors from its classifying category to **set**. Such functors will necessarily carry the object representing a query to the set of records which satisfy the query.

4 Constraint specification

We show by example how to model the vast majority of the integrity constraints required in information systems by using ER-modelling with commutative diagrams. Some examples of the constraints which can be treated include the requirement that in a database of students, courses, classes and class times, it is required that no student have a clash between two timetabled classes; when

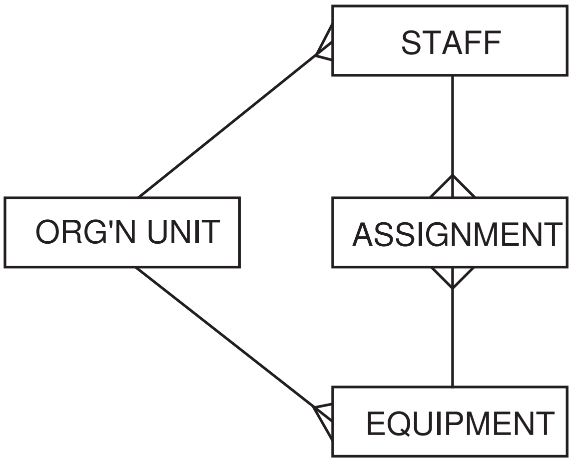

Figure 3: An information model with a single static constraint (the diagram is required to commute)

an order is delivered it must be delivered to the address of the customer who placed the order; and when a contractor engages in some work involving a business resource there must be a contract that specifies that that contractor has the right to use that resource.

Some complicated constraints require the use of the query language outlined above (since a constraint may apply only to certain entities determined by a description that can be used as a query).

4.1 Static constraints

A static constraint is a constraint which must be satisfied throughout the life of the information system. Such constraints may be easily modelled using commutative diagrams and they are very common. As an illustrative example consider the model shown in Figure 3. The model involves members of staff and pieces of equipment, both of which belong to some organisational units within a business. Pieces of equipment may be assigned to the use of one or more members of staff. It may be a rule of the business that a piece of equipment can only be assigned to a member of staff if they are both part of the same organisational unit. This rule may be enforced simply by requiring the diagram to commute. Of course, if there is no such rule then the diagram may or may not commute depending on whether the assignments happen to "accidentally" only assign equipment to staff when the equipment and the staff belong to the same organisational unit.

Now that we have seen how static constraints are modelled by commutative diagrams, it is appropriate to return to Figure 2 and to see how the search for commutative diagrams can improve the resulting information model (as promised in Section 3.1).

The model shown in Figure 2 was completed before we had realised the importance of commutative diagrams. Nevertheless it contains many static constraints, one example being a four-sided commutative diagram that ensures that every export movement happens under a contract involving a company (as supplier) which must be the same company which owns the stockpile being moved for export. Furthermore the search for commutative diagrams reveals possible improvements of the model: It is surprising that the entity LOADING is not involved in a commutative diagram with some contractual entity. In fact there should be a many-to-one relation between LOADING and SHIP CONTRACT which says that each loading onto a ship occurs under a ship contract that involves that ship. In addition, that ship contract indicates under which (export) contract the export is to take place. However, in order to obtain the commutative diagram which will ensure that the (export) contract is the very contract which resulted in the export stockpile we must move the relation between EXPORT MOVEMENT and CONTRACT so that it goes between EXPORT STOCKPILE and CONTRACT. This results in no loss because the EXPORT MOVEMENT to CONTRACT relation can be recovered by composition, and it permits the important constraint (commutative diagram) that the composite from LOADING to EXPORT STOCKPILE to CONTRACT is equal to the composite LOADING to SHIP CONTRACT to CONTRACT.

4.2 Dynamic constraints

There are many examples of business rules and government regulations which provide constraints which are variable. For example there are work practices which have required that only a certain class of employee can engage in certain classes of work but which are now being liberalised with industrial relations reforms which are underway in a number of countries. To accommodate these *dynamic constraints* we must have an entity which embodies the constraint information, since only instances of entities can be created and destroyed in an information system.

An extension of the previous example which incorporates a dynamic constraint appears in Figure 4. Once again staff may be assigned the use of pieces of equipment, but if we require the two triangles to commute then there must have been a prior authorisation which said that that particular assignment was permitted. Thus, if staff are only allowed to use equipment that they have been trained to use then when a person attends a training session for a particular piece of equipment there will be an entry placed in AUTHORISATION recording that they are permitted to use that equipment. They may never be assigned that piece of equipment, but if they are then the relationship between ASSIGNMENT and AUTHORISATION will refer back to the entry made after their training.

Interestingly there may also be higher level constraints implied here. If the static constraint described in Section 4.1 still applies here, then since the left quadrilateral and the two triangles commute, the outside quadrilateral must commute too. But this constrains possible authorisations: A staff member may only be *authorised* to use a piece of equipment if that staff member and that piece of equipment belong to the same organisational unit.

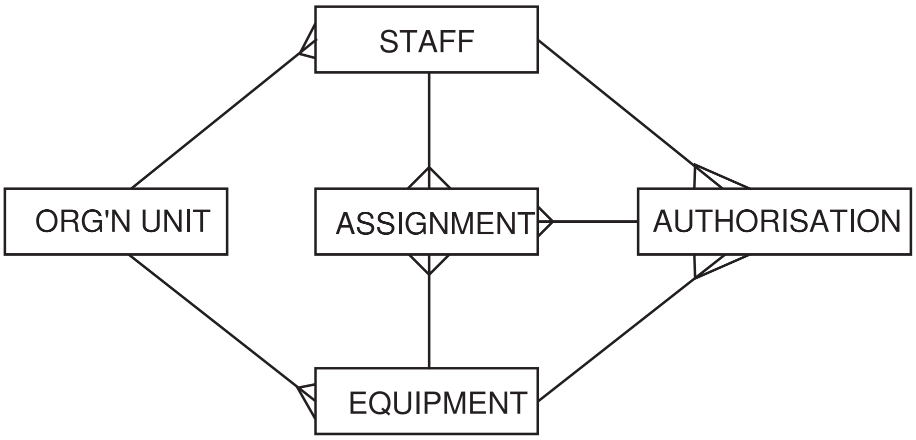

Figure 4: An ER fragment with a dynamic constraint (the two triangles on the right hand side are required to commute)

4.3 Constraints and queries

Some constraints are so specific as to involve queries. For example the constraint that in a database of students, courses, classes and class times, it is required that no student have a clash between two timetabled classes, it is necessary to proceed as follows (Figure 5). First since the relationship between students and courses may be many-to-many we introduce a new entity called ENROLMENT. Each instance of ENROLMENT indicates that a particular student is enrolled in a particular course. We assume that each course has several classes each with a time slot. If we form the pullback of the arrows from ENROLMENT and CLASS into COURSE then what we obtain is essentially a time-table: Instances of the pullback may be thought of as triples (student, course, class) where the student is enrolled in the course which has as one of its classes the class. Of course there is a map from this TIME-TABLE entity to the product of STUDENT and TIME SLOT. To satisfy the constraint we simply require that this map be monic (which of course can be stated in a Boolean category).

5 Process modelling

Once a satisfactory ER-model has been developed it is common to work out a process model for the business which shows the important processes carried out by the business and how they trigger one another. The process model will be much less general than the ER-model since it will tell us about how the business is currently organised (and this may change despite the information needs of the business remaining the same).

Traditionally the process model is influenced by the ER-model, but our new methodology for ER-modelling makes the link explicit. Consider the diagrams in the ER-model which have been specified as commuting. Typically each of

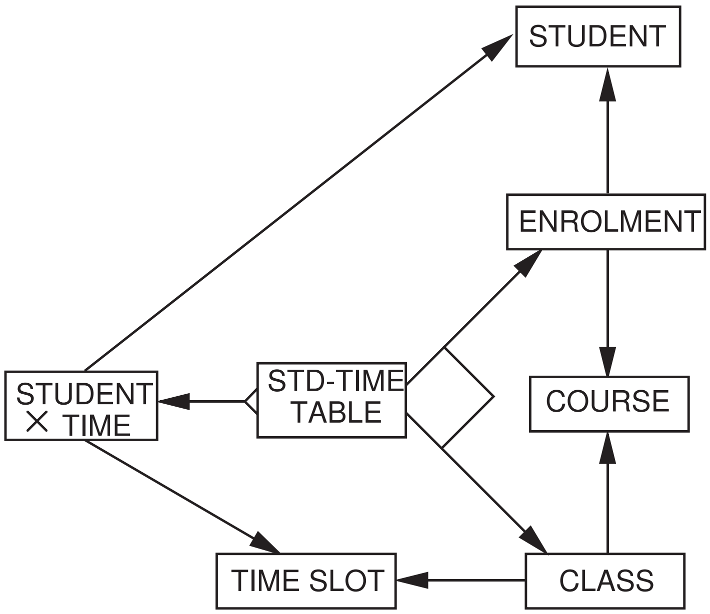

Figure 5: A student-class information model expressed in categorical (rather than ER) notation. The requirement that the central horizontal arrow be monic expresses the "no clashes" constraint.

these loops represents an individual process and reconciliation cycle. This is because, in order to update the information system, it is usually necessary to update an instance at each vertex of the diagram and then finally to check that commutativity has been preserved (see Figure 6).

Thus to develop the process model one calculates a kind of graph dual of the ER-model in which specified commutative diagrams correspond to processes and common vertices between such commutative diagrams correspond to triggers between the processes.

Of course it is often the case that an analyst can further refine the process model, but it is useful to note that the greater part of the work of developing a process model has already been done if one has specified the commutative diagrams in the ER-model.

It should be noted that not all process and reconciliation cycles are as simple as those suggested by Figure 6. Figure 7 shows the processes cycle involved in preparing a loan in a bank process model.

6 Views and architectural implications

The methodology that we have been describing also has some as yet untested implications for other aspects of information system development.

One particularly difficult problem in dealing with large information systems is the presentation of different *views* of the system for different users. The problem is essentially one of how to partition the system so that users can see a relatively complete view related to the aspects that are of relevance to them without having to look at the whole system. The recognition of commuting diagrams as processes suggests that the best partitioning would be obtained by choosing a related group of commutative diagrams. This will be developed in work currently in progress. An early example is shown in Figure 8 where four views of an information model have been derived by selecting commuting diagrams from a much larger and more detailed model.

This partitioning can be carried further. The growth of very large information systems has led to problems of complexity and context retention which might best be solved by allowing business units a certain autonomy with their information systems. However, integration of such systems is necessary and the complexity of the interaction between subsystems can be dangerous. We propose the development of a corporate information (ER) model which can be used to determine, via commutative diagrams how to partition the system into subsystems. This will require duplicating entities that happen to fall into two subsystems and providing a message passing mechanism to allow the two copies to remain synchronised. However, if the partitioning is done well, and we believe an analysis based on commutative diagrams will do this, then it is likely that the interaction between subsystems will be quite manageable.

This proposed architecture for information systems is the source of the name Federated Information Systems.

An example of a federated information system appears in Figure 9

References

[1] M. Barr and C. Wells, *Category theory and computer science*, Prentice Hall, 1990.

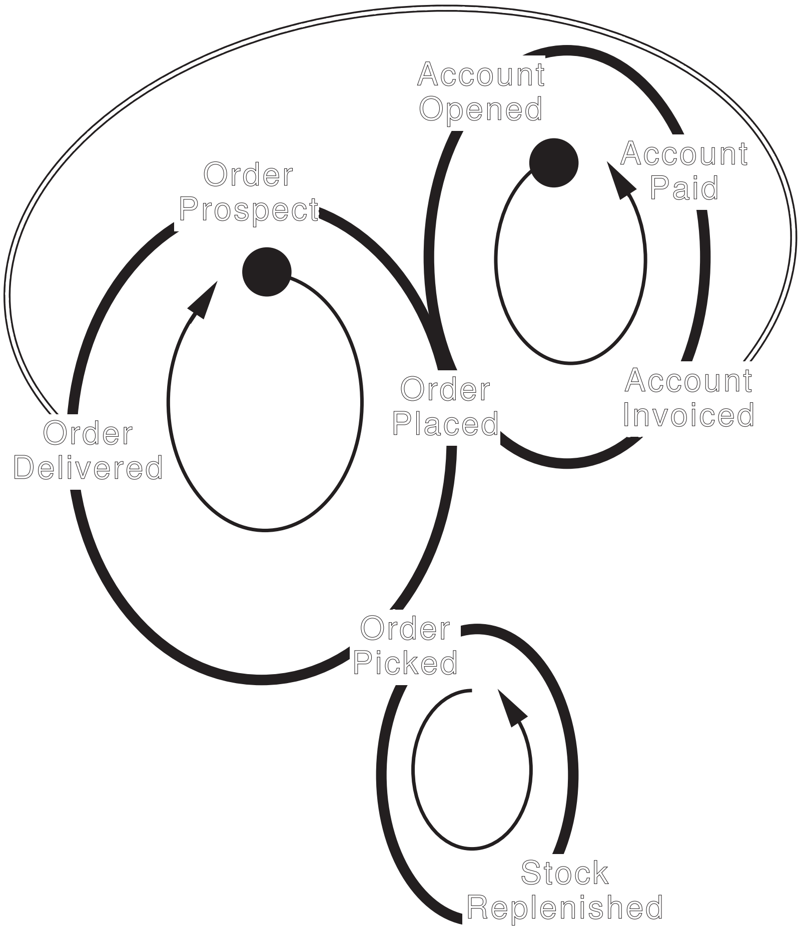

Figure 6: An example of interacting process and reconciliation cycles (the double lines from Order Delivered to Account Invoiced represent a trigger—we assume that an account is not to be invoiced until the corresponding order has been delivered)

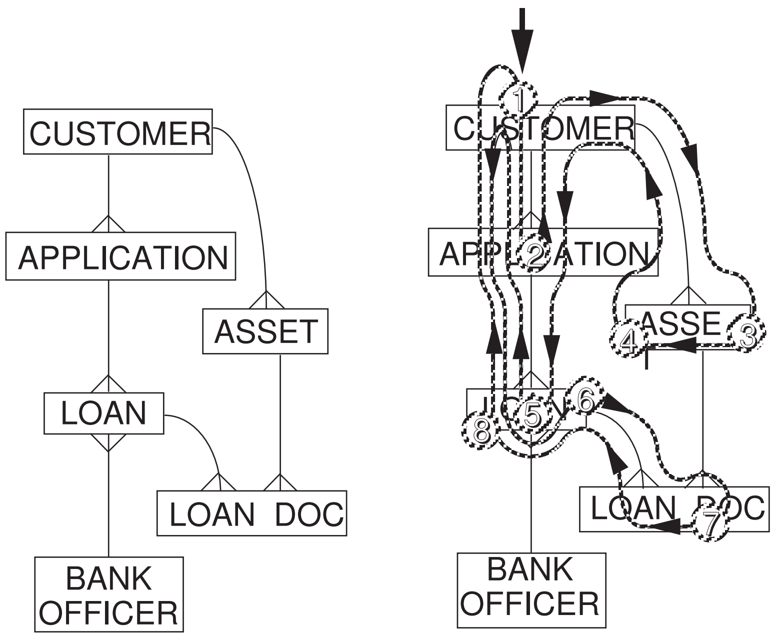

Figure 7: A fragment of a banking information model and its process cycle. The numbered processes are: (1) Check identification, (2) Initiate application, (3) Appraise assets, (4) Check title to assets, (5) Check previous loans (credit history), (6) Gain bank officer approval, (7) Finalise loan documents, (8) Complete loan.

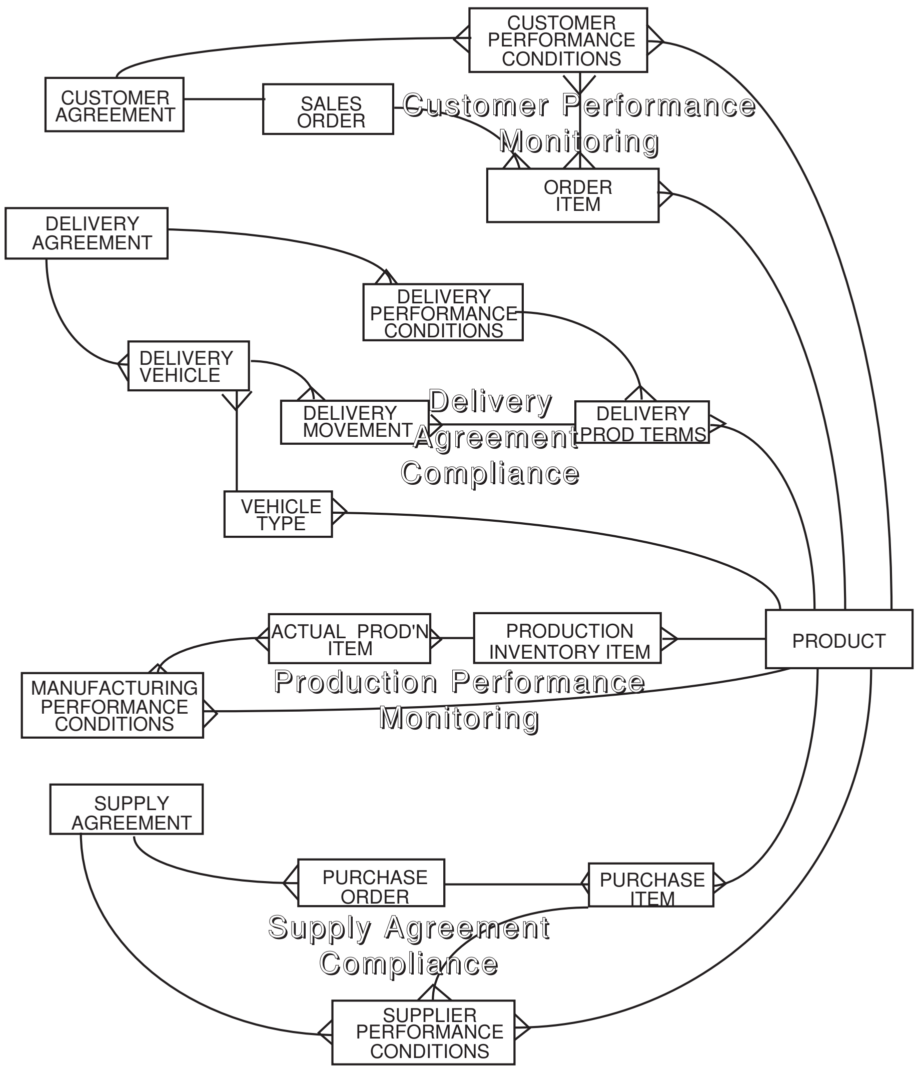

Figure 8: Four views extracted from a large information model by selecting commuting diagrams

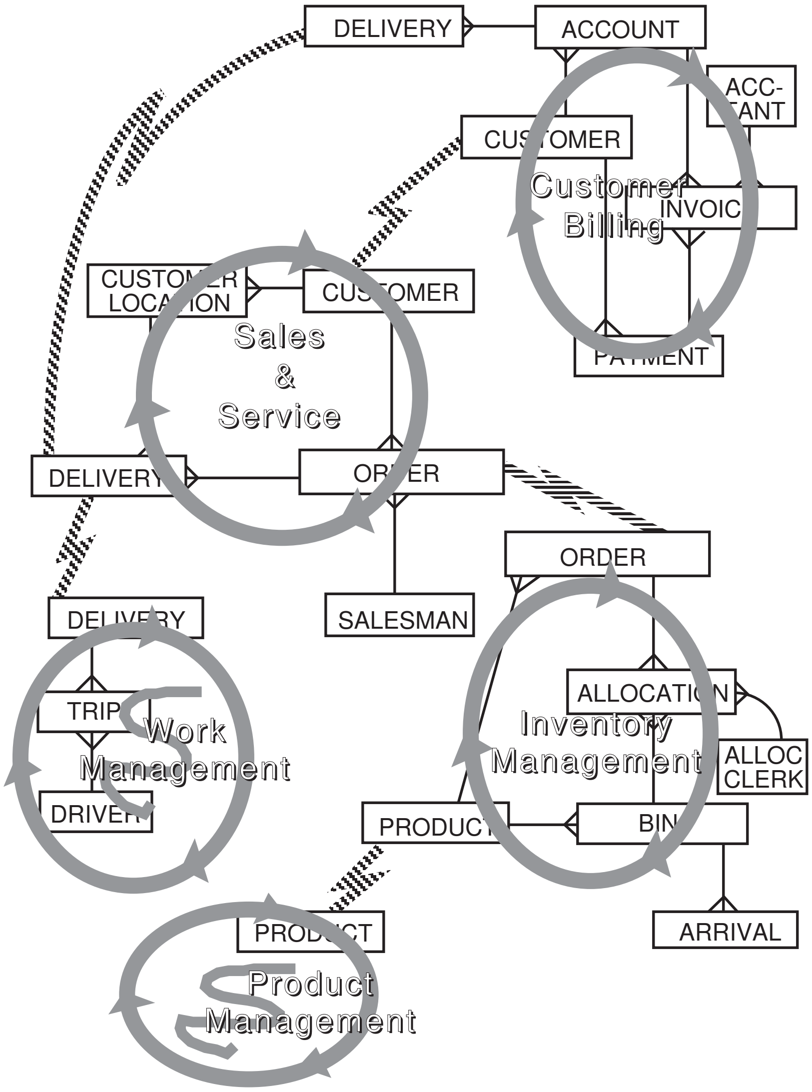

Figure 9: A fragment of a federated information system

[2] A. Carboni, S. Lack and R.F.C. Walters, Introduction to extensive and distributive categories, *Pure Mathematics Report* 92-9, University of Sydney, 1992.

[3] P.P.S. Chen, The entity relationship model—towards a unified view of data, *ACM Transactions on Database Systems*, **1** (1976), 9–36.

[4] C.N.G. Dampney, M. Johnson and P. Deuble, Taming large complex information systems in D.G. Green and T. Bossomaier (eds) *Complex Systems* IOS Press, Amsterdam, (1993), 210–222.

[5] C.N.G. Dampney, M. Johnson and G.P. Monro, An illustrated mathematical foundation for ERA, in C.M.I. Rattray and R.G. Clarke (eds) *The unified computation laboratory*, Oxford University Press, Oxford, (1992), 77–84.

[6] C.A. Gunther, The mixed power domain, to appear in *Theoretical Computer Science*.

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

[8] R. Rosebrugh and R.J. Wood, Relational databases and indexed categories, *Canadian Mathematical Society Conference Proceedings* **13** (1992), 391–407.

[9] S. Vickers, Geometric theories and databases, *London Mathematical Society Lecture Notes* **177** (1992), 288–314.

[10] R.F.C. Walters, *Categories and computer science*, Cambridge University Press, 1992.