Acta Informatica 38, 1–44 (2001)

© Springer-Verlag 2001

Category-theoretic fibration as an abstraction mechanism in information systems

Robert M. Colomb[1,⋆], **C.N.G. Dampney**[2], **Michael Johnson**[2]

[1] School of Computer Science and Electrical Engineering, The University of Queensland, St. Lucia, Queensland 4072 Australia (e-mail: colomb@csee.uq.edu.au)
[2] Department of Computing, Macquarie University, NSW 2109 Australia

Received: 23 June 1997 / 1 June 2001

Abstract. This paper examines the problem of establishing a formal relationship of abstraction and refinement between abstract enterprise models and the concrete information systems which implement them. It introduces and justifies a number of reasonableness requirements, which turn out to justify the use of category theoretic concepts, particularly fibrations, to precisely specify a semantics for enterprise models which enables them to be considered as abstractions of the conceptual models from which the implementing information systems are built. The category-theoretic concepts are developed towards the problem of testing whether a system satisfies the fibration axioms, and are applied to case studies to demonstrate their practicability.

1 Introduction

An enterprise model [e.g. 12] represents an enterprise's information structure as a whole. It functions as a framework for the various information systems which support specific business processes which in turn implement the information flows. These models are typically very abstract – they represent on a single sheet a corporate information system which which when it is refined might involve hundreds of entities and relationships. Other workers [e.g. 3] create a similar abstract model as a documentation exercise, to attempt to gain an overview of a conceptual model or as a top-down method of constructing a conceptual model. In this paper, we will use the term "enterprise model" to denote the high-level abstract model, and the term "imple-

⋆ Work performed partly while visiting Department of Computing, Macquarie University

mentation models" to denote the conceptual models directly supporting the organization's information systems. Furthermore we consider the totality of the implementation models so that the enterprise model functions not only as a framework but as a *template* for the implementation models.

Conceptual models based on the entity-relationship or similar approaches are valuable since

- they are based on a precise mathematics,
- they model a reality as it is perceived by people, and
- they are very closely related to the database implementation of the information system.

Their main disadvantage is their size- they may have hundreds of entities and relationships- so they are difficult to comprehend as a whole. People who need an overview are not well served by these conceptual models. Ideally, these people should be served by enterprise models. An example of an enterprise model is given in Fig. 1. The model is expressed in the same notation used in entity-relationship modelling, but the notation has a much weaker, more general, semantics.

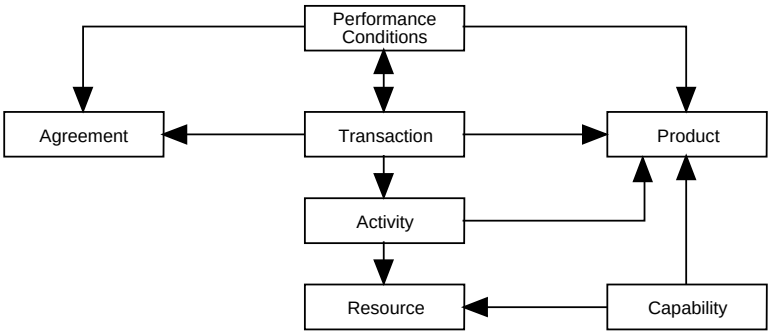


Fig. 1. An abstract enterprise model

An entity in an entity-relationship model represents a set of objects, which generally corresponds to a table in the database. An abstract entity in an enterprise model generally represents a large segment of the organization's information system. A relationship in an entity-relationship model represents dependencies among entities, and these dependencies have a semantics derived from the theory of functions. An abstract relationship in an enterprise model expresses that the abstract entities are somehow related. There is no accepted and definitive semantics for the many-to-one and many-to-many symbols on the diagram. The problem is that although the enterprise model provides higher management with an overview of the organization, it is difficult to tell how the overview relates to the organization's actual

information flows, so that the enterprise model has limited usefulness for determining requirements and for deployment.

A second class of person who needs an overview are the software engineers. Hierarchical decomposition is the main way that software engineers control complexity. A high-level diagram of a system allows the engineers to see for example what parts are likely to be affected by a proposed change. Also, system design and construction often proceeds by successive refinements of abstract diagrams. Some workers employ abstract schemas to achieve this kind of overview of the conceptual models they construct and manage [3]. An example of an abstract schema is shown in Fig. 2. These abstract schemas have the same difficulties as abstract enterprise models: although the notation they use is similar to entity-relationship notation, it lacks semantics so that the abstract schemas are difficult to relate to the detailed conceptual models, and are of limited utility.

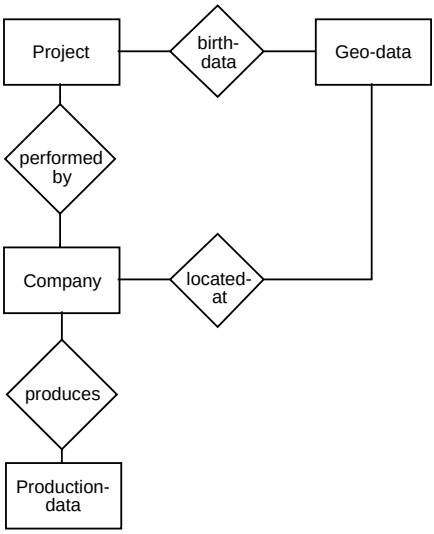


Fig. 2. An abstract schema (Source: [3], p. 175)

This paper first establishes a set of plausible requirements that should be satisfied for

- enterprise model schema, together with
- the abstraction-refinement correspondences from an implementation model to an enterprise model and vice-versa.

This shows that the category theory ideas surrounding fibration provide a natural formalisation of the requirement.The fibration construct is examined in depth, with a number of examples provided to show the consequences of

the requirements, and a number of propositions demonstrated which simplify the task of testing whether a candidate abstraction in fact satisfies the requirements.

The theory is then applied to an extensive example, showing that the large scale integrity constraints implied by the enterprise model have useful consequences for the implementation model, and that the fibration abstraction mechanism is practicable. Finally, satisfaction of some of the abstraction requirements needs additional structure provided by fibration, namely that the fibration has a right inverse with certain properties. The consequences of these additional axioms are explored with examples and propositions, then applied successfully to the case study. The conclusion section summarizes the lessons learned, and discusses some interpretations of the category-theory structures.

2 Requirements for an enterprise model to be an abstraction

We will focus our attention on the enterprise data model of Fig. 1. What does this tell us about the information systems upon which the operation of the enterprise depends? The notation used is the same as is used in entity-relationship modelling. There are two blocks labelled respectively *Transaction* and *Product*. They are connected by a line annotated as a many-to-one relationship. If this were an ER model, the line would denote that each instance of *Transaction* is associated with at most one instance of *Product.* Similarly, the line between *Transaction* and *Activity* would denote that each instance of *Transaction* was associated with at most one instance of *Activity.* Further, we could infer that each instance of *Transaction* was associated with at most one instance of *Product* in two different ways: once directly and once indirectly through *Activity*. We describe this constraint – that the dependence on *Product* is the same in both cases – as *consistent dependency*. This interpretation of the ER notation depends on the interpretation of entities as sets of instances and of many-to-one relationships as functional dependencies.

Note, by the way, that it is possible to assume that in a conceptual model every many side of a many-to-one relationship is mandatory. If necessary, we can introduce a subtype structure to achieve this. Furthermore, it is possible to assume that there are no many-to-many relationships in the ER model. If necessary, we can achieve this by introducing new association entities. Similarly, we can assume that all relationships are binary. Therefore, we can can without losing generality assume that all the relationships in the entity-relationship model designate total functional dependencies. That is, all relationships are binary and mandatory on the many side.

Let us now look at a fragment of an implementation model of the information system for the same organization as the enterprise model of Fig. 1, expressed in a modified ER notation, in Fig. 3. A relationship is represented by an arrow. Only many-to-one relationships are permitted. The *one* side has an arrowhead, the *many* side does not. A mandatory relationship on the one side is represented by a large dot next to the arrow, so that for example every *Vehicle Type* has at least one *Transport Capability* associated with it. Similarly *Order Item* has at least one *Customer Performance Condition*.

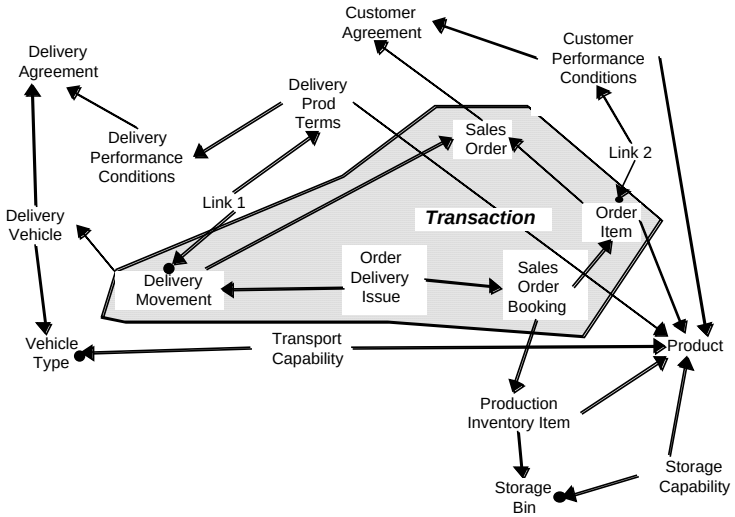


Fig. 3. An information system implementation model for the organization of Fig. 1

The block *Product* of the enterprise model corresponds to the entity *Product* of the implementation model, so that we are justified in taking the enterprise model object *Product* as an entity. However, the block *Transaction* in the enterprise model is intended to correspond to the five entities *Sales Order*, *Order Item*, *Sales Order Booking*, *Order Delivery Issue* and *Delivery Movement*. We could treat the enterprise model block *Transaction* as a generalization of the implementation model entities.

To do this using set theoretic reasoning every instance of every entity of the implementation model must correspond to an instance of an entity of the enterprise model and every existing instance in the enterprise model must correspond at least to an instance of an entity in the implementation model. The reader will note that in set theoretic terms we have sets in which we allow new entity instances to come into existence. The technical difficulties that this causes in set theory is one good reason to use category theory instead.

We could treat the enterprise model block *Transaction* as some kind of generalization of a composition of implementation model entities. The

composition involved in the mapping between the enterprise model and the implementation models also suggests some kind of aggregation of implementation entities.

Exactly how the mapping is defined in terms of conventional generalisation or aggregation used set theoretically in extended entity-relationship modelling is not critical for the solution we have in mind. The solution, in keeping with category theory approaches, is based instead on a mathematical construct determined by the external relationships of the entities and blocks, not the internal structure of entity instances.

We will refer to the entities in the implementation model corresponding to an entity in the enterprise model as being *above* the enterprise model entity. We will call this requirement **R1.**

R1a: Every entity in the implementation model is above an entity in the enterprise model (abstraction), and
R1b: Every entity in the enterprise model has at least one entity in the implementation model above it (refinement).

We now consider the interpretation of the connecting lines in the enterprise model. At least, we would like to say that if there is no relationship between two entities A and B in the enterprise model, there should be no relationship between any of the entities in the implementation model above A and any of the entities in the implementation model above B. The notation of the enterprise model suggests a direction of relationship, and the semantics of the ER notation in the implementation model is directional, so it makes sense further to require that the directions correspond. If we do not require this, it is hard to see how the enterprise model can be considered an abstraction of the implementation model. Such a requirement is part of other widely-used abstraction mechanisms, notably *data flow diagrams* [7]. This issue leads to requirement **R2:**

R2: If there is a relationship in the implementation model between two entities X and Y, then there is a relationship in the enterprise model between the entities A and B, where X is above A and Y is above B. Moreover, transitive relationships in the implementation model are preserved in the enterprise model.

We will say that if there is a relationship f in the enterprise model between entities A and B, then any relationship between X above A and Y above B (in the same direction as f) is *above* f. Again, if we want the implementation model to be a refinement of the enterprise model, it makes sense to require **R3**:

R3: If there is a relationship in the enterprise model, there must be at least one relationship in the implementation model above it.

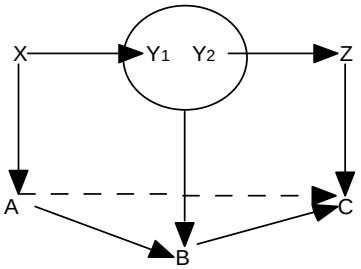


Fig. 4. Example fails transitivity

The semantics of the relationships in the ER model is functional dependency. Therefore, in the implementation model if there is a many-to-one relationship between X and Y and a many-to-one relationship between Y and Z, there is a derived many-to-one relationship between X and Z. Note that we want R3 to apply to derived relationships as well as the primary relationships in the implementation model.

If we want the enterprise model notation to have this derived relationship semantics, then we must place some further restrictions on the relationships we expect in the implementation model. For example, consider Fig. 4.

The lower model ABC is an enterprise model, and the upper model XY_1Y_2Z is an implementation model which satisfies R1, R2 and R3. There is a derived relationship $A \rightarrow C$ in the enterprise model, but no relationship between X and Z in the implementation model. To insure that this transitive behaviour in the enterprise model is preserved in the behaviour of the implementation model (refinement), we need requirement **R4**:

R4: If there is a relationship in the enterprise model, and there is a derived relationship obtained by composing a second (*connecting*) relationship with the first, then there must be a relationship in the implementation model above the derived relationship which is derived from relationships above the first and connecting relationships.

The principle of consistent dependency applies in the implementation model. This principle is exemplified in the model of Fig. 3. Consider the entity *Sales Order Booking*. It has two indirect dependencies on *Product*: one via *Order Item* and the other via *Production Inventory Item.* An instance of *Sales Order Booking* is an assignment of a particular storage bin to an order item. The principle of consistent dependency is an integrity constraint which in this instance requires that the product reached via either route is the same. In other words, the product in the bin is the same product as the one in the order. This constraint is not implied by the usual constraints on the ER diagram, so must be specified by the systems designer. Of course all

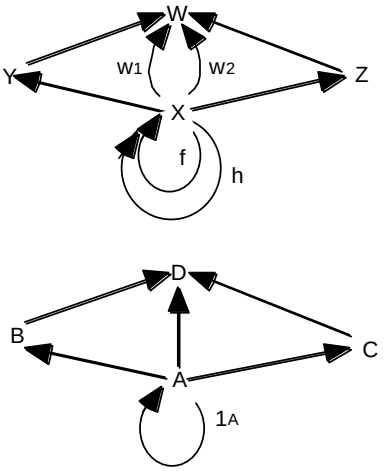


Fig. 5. Consistent dependency in enterprise and implementation models

derived relationships must satisfy consistent dependency. The principle is discussed in more detail in e.g. [5].

For the enterprise model to have the same semantics, and for the implementation model to be a refinement of the enterprise model, we need requirement **R5**

R5: If a set of relationships in the enterprise model satisfies the principle of consistent dependency, then every set of relationships above them must also.

R5 is actually a consequence of R4. Consider the models in Fig. 5.

If the indirect relationship A $\rightarrow$ D via B is the same as the indirect relationship via C, the enterprise model satisfies the principle of consistent dependency. It satisfies R4 for there to be two relationships between X and W; w_1 satisfying consistent dependency with X $\rightarrow$ Y and Y $\rightarrow$ W, w_2 satisfying consistent dependency with X $\rightarrow$ Z and Z $\rightarrow$ W. However, we can assume that every entity participates in an identity relationship (a relationship associating each instance of the entity with itself). The identity relationship for A is denoted by 1_A. The relationships A $\rightarrow$ D and (the somewhat artificial but quite valid) A $\rightarrow$ D via 1_A satisfy consistent dependency. By R4 there must be an f above 1_A such that w_2 is the composite of w_1 following f. The implementation relationship f need not be the identity; it could for example be a permutation of X. Note that the same argument shows that there must be a relationship h above 1_A such that w_1 is the composite of w_2 following h. So long as h followed by f is the identity, and vice versa, it seems reasonable to accept this situation as satisfying R5.

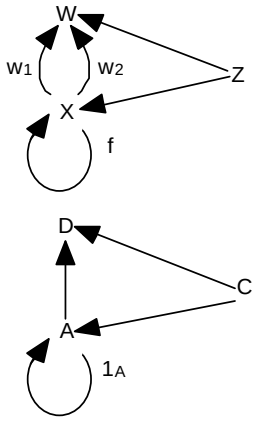


Fig. 6. Multiple solutions to R4

A concrete example of this situation is as follows. Take X to be a set of staff members and W to be a set of responsibilities. Each staff member has a primary responsibility (w_1) and a secondary responsibility (w_2). Each staff member has also a deputy among the staff members (f), and the deputy has the reverse responsibilities. In this way w_2 is f followed by w_1.

Further considering R4, the requirement permits there to be possibly several distinct connecting relationships. In Fig. 6, for example, we could have both w_1 and w_2 following $Z \rightarrow X$ being the same as $Z \rightarrow W$. An argument similar to that used in R5 shows that w_1 and w_2 must be related by an f and h in the same way. This leads to a further refinement of R4, Which we will call **R6:**

R6: The solutions to R4 must be equivalent.

(In category theory, isomorphic objects are indistinguishable, in that they form equivalence classes. If one member is the source or target of an arrow, then so are all the other members. Of course, there can be multiple isomorphisms among the objects, as f and h of Fig. 5.).

We want the enterprise model to somehow characterize the information system. Suppose there is a relationship f from A to B in the enterprise model, and that there are several entities in the implementation model above both A and B. Requirement R3 has it that there is at least one relationship in the implementation model above f. However, suppose that most of the entities above A and B do not participate in relationships above f. We argue that this situation is unsatisfactory, and that the resemblance between the enterprise and implementation models should be closer for the enterprise model to be useful as an abstraction of the implementation model. Further, consider an enterprise model containing $A \rightarrow B \rightarrow C$, with the derived $A \rightarrow C$. R3 entails that every implementation model entity above B which participates in

a relationship above A → B also participates in a relationship above B → C. We therefore already have some additional, somewhat ad hoc, constraints on the implementation model. It would be more satisfactory to have a stronger axiomatic requirement which is more universal. Such stronger requirements are given in **R7B** and **R7F**:

R7B: Every entity above B must participate in a relationship above f.
R7F: Every entity above A must participate in a relationship above f.

Finally, we might want to say that the enterprise model is a template for the actual relationships existing in the implementation model. That is, if there is a connected structure in the enterprise model (the whole of Fig. 1 is a connected structure), then there ought to be implementation entities above the enterprise model entities, and implementation relationships above the enterprise model relationships, which have the same pattern.

R8: At least one implementation entity can be selected above each enterprise model entity such that they participate in the same pattern of relationships as the enterprise model.

This leads finally to the requirement which at this stage we state informally as the meaning of the word *appropriately* in the following statement is not yet fully specified:

R9 (informal): Every entity in the implementation model participates *appropriately* in such a pattern.

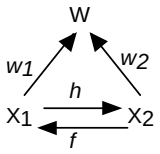
3 Category theory and information systems

It has been established that a conceptual model in the ER formalism satisfies the axioms to be a category, in the theory of categories [5], [9], [10] (For a general introduction to the theory of categories, see e.g. [1]). A category is a system consisting of objects (from here on the word object refers to object as defined in category theory) and arrows. An arrow f : A → B has one object A as its source and another B as its target. In our application, an entity is denoted by an object, and a relationship by an arrow. The (mandatory) many side of the relationship is the source of the arrow, and the one side is the target of the arrow.

To be a category, the objects and arrows must satisfy three axioms: transitivity, associativity and identity. *Transitivity* means that if f:A→B and g:B→C are arrows, so is gf:A→C. *Associativity* means that the sequencing of composition makes no difference. Thus if h:C→D hgf = h(gf) = (hg)f. *Identity* means that for every object A, there is an arrow 1_A:A→A such that for every f and B with f:A→B, $f1_A = f$, and $1_B f = f$. In the ER modelling

conventions used in Fig. 2, every entity is assumed to be both the source and target of an identity arrow, and we already have that the arrows are transitive. One notes that these axioms are very general indeed. Further axioms are added to this basic collection which define additional constructs enabling more specific theory to be developed.

In category theory, an object is viewed entirely as a black box. The focus of the theory is mostly on arrows between objects, usually expressed in graph form (nodes and directed edges) with the graph and its sub-graphs referred to as external diagrams, in this paper simply *diagrams*. Arrows can be interpreted as specifications which the objects must satisfy in any interpretation. The content of an object is therefore abstracted away, except that the internal structure of an interpretation of an object must be sufficient to satisfy the specifications embodied in the arrows with which it is associated. For this reason, category theory uses *isomorphism of objects*, which is a weak concept of equality. Consider



The objects X_1 and X_2 are isomorphic and thus indistinguishable from the point of category theory if $hf = id_{X1}$ and $fh = id_{X2}$. Informally we might say that the two objects are the *same*. This equality between arrows is a main concept in category theory. The two arrows, possibly derived by composition, have the same source and target. Their equality may be either assumed or proved and they are said to *commute*. The general structure of proofs in category theory is that some arrows in a diagram are assumed to commute, and therefore other arrows can be proved to commute. (If all arrows in a diagram which could commute in fact do so, the diagram is referred to as a *commuting diagram*.)

At this point, we should pause to clarify what we are modelling. The issue addressed in this paper is the overall structure and strategic control of an organization's information system. The data structures and constraints in the system are modelled by enterprise models and by implementation models (Figs. 1 and 3, respectively). The information system itself is implemented as a database system, which consists of a schema and a population. There are well-known algorithmic techniques for constructing database schemas from conceptual models e.g.[6]. The population of a database is established by a sequence of updates, and queries on the database are specified using the relational algebra or one of its equivalents such as SQL.

We are taking the implementation and enterprise models as categories. Since, however, there is an algorithmic correspondence between conceptual models and databases, we can say that the databases are logical models of the categories even though our work is focussed on the conceptual models.

Several authors have developed category theoretic formulations of relational databases. Rosebrugh and Wood [11] model the relational algebra as a monoid. Their formulation includes all relational and set-theoretic operations (including updates), except set difference. Islam and Phoa [8] model the relational system using Scott domains as value sets for attribute functions. Baclawski *et al.* [2] use internal categories to separate schema from database state. It seems clear from these works that a complete formulation of the relational database can be modelled in category theory. It is useful to know that the database aspect of information systems can be modelled using category theory, but it is not necessary to do so for our purposes: viewing the implementation and enterprise models as categories is sufficient.

The principal interpretation of the categories under discussion will be as subcategories of the category whose objects are finite sets and whose arrows are total functions, called *FinSet*. Two arrows commute if they represent the same function. Interpreting the conceptual models as subcategories of FinSet means that we model a database at the level of functional dependencies, which abstracts away from particular representations in the form of tables. Functional dependencies are also the level at which much of the theoretical work in relational database is carried on. The category FinSet has a number of derived structures which we will sometimes need to import into our models.

4 Application of category theory to the abstraction problem

The category theoretic concept implementing the requirements we have established for the abstraction problem is that of fibration, which was first proposed in [4]. This section provides the necessary definitions to formalize the requirements.

4.1 R1-3 and the concept of surjective functor

Since both the implementation model and the enterprise model satisfy the category axioms, they are both categories. Call the implementation model C and the enterprise model E.

Requirements R1 and R2 in category theory are encapsulated by the concept of a *functor* $P : C \rightarrow E$. Requirement R3 in category theory is that the functor must be *surjective on both objects and arrows.*

A functor P is a graph homomorphism from the objects and arrows of category C to the objects and arrows respectively of category E, subject

to the constraints that identities are preserved and if f:A→B and g:B→C are arrows of C, then PgPf:PA→PC in E = P(gf). (The mapping preserves transitivity). An example of a functor is given in Fig. 7.

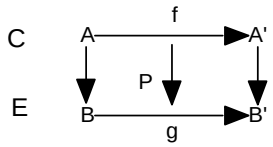


Fig. 7. Example of functor P:C→E

Some detail can be lost, as in Fig. 8, where f is carried to the identity on B.

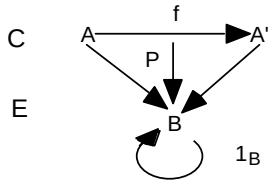


Fig. 8. Functor P:C→E preserves structure but loses detail

The graph homomorphism itself satisfies R1, while the transitivity-preserving property of a functor satisfies R2.

4.2 R4-6 and the concept of cartesian arrow

Requirements R4, R5, and R6 are formalized by the concept that a relationship is a *cartesian* arrow. The notation Obj(C) refers to the set of objects in the category C, and Arr(C) refers to the set of arrows in the category C.

Let P:C → E be a functor
X in Obj(C) is **above** A in Obj(E) if PX = A
c in Arr(C) is **above** f in Arr(E) if Pc = f

In Fig. 9, Object X is above A, Y is above B and Z is above C. Arrow c is above f, w is above h, and v is above fh.

Further, c:Z→Y is *cartesian* with respect to a given P (for f and Y), if Pc = f and

$$\forall\ v{:}X \to Y \text{ in } \mathrm{Arr}(\mathsf{C}) \text{ and } h{:}PX \to C \text{ in } \mathrm{Arr}(\mathsf{E}) \text{ such that } fh = Pv$$

$$\exists!\ w{:}X \to Z \text{ in } \mathrm{Arr}(\mathsf{C}) \text{ with } wc = v \text{ and } Pw = h$$

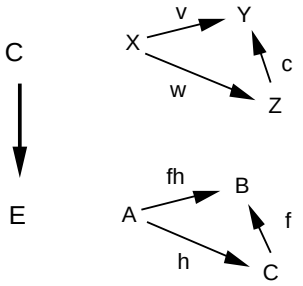


Fig. 9. Structures used in defining a *cartesian arrow*

R4 can be implemented by the requirement that there be a cartesian arrow above every arrow of E. Therefore, if there is a commuting diagram in E, there must be at least one commuting diagram in C which is carried into the diagram in E. Furthermore, for every arrow of C carried into the composite arrow of E, there is a unique connecting arrow in C completing a commuting diagram which is carried into the commuting diagram in E. (The *for every arrow of* C clause is similar to R5, while the *unique connecting arrow in* C clause is similar to R6.)

4.3 R7B and the concept of fibration

The concept of a cartesian arrow, together with R7B, can be formalized by the functor being a *fibration*. A functor P is a *fibration* if $\forall$Y in Obj(C) with PY = B and $\forall$f:C $\rightarrow$ B in Arr(E) $\exists$ cartesian c for f and Y in Arr(E) above f. Intuitively, every arrow in E whose target is the image of an object in C is the image of a cartesian arrow in C, for each object of C carried into the target in E. This is equivalent to requirement R7B. In Fig. 9, we have c the cartesian arrow for f and Y, w the cartesian arrow for h and Z, and v the cartesian arrow for fh and Y. The composite arrow associated with h is $h1_A$. Similarly, the composite arrow associated with fh is $fh1_A$. Associated with f are the composites fh and $f1_C$. The concept of fibration is analysed below, where a number of examples are given to help build intuition, and a number of propositions proved which make the concept easier to apply.

4.4 R7F and the concept of opfibration

R7B requires that every entity carried into the target of an enterprise relationship have a certain property, leaving the entities carried into the source of the enterprise relationship relatively unconstrained. R7F is a similar requirement on the entities carried into the source of the enterprise relationship, leaving the entities carried into the target relatively unconstrained. Category

theory has the concept of *opfibration*, dual to fibration, which is equivalent to R7F.

We need a concept of *opcartesian arrow*, dual to cartesian arrow. A cartesian arrow has a property for every connecting arrow composing on the right. An opcartesian arrow has that property for every connecting arrow composing on the left. In Fig. 9, arrow v is opcartesian for h and X.

A functor P is a *opfibration* if $\forall$X in Obj(C) with PX = A and $\forall$h:A $\rightarrow$ C in Arr(E) $\exists$ opcartesian w for h and X in Arr(E) above h. Intuitively, every arrow in E whose source is the image of an object in C is the image of an opcartesian arrow in C, for each object of C carried into the source in E. This is equivalent to requirement R7F. In Fig. 9, w is the opcartesian arrow for h and X, and v the opcartesian arrow for fh and X. The composite arrow associated with f is 1_{B}f. Similarly, the composite arrow associated with fh is 1_{B}fh. Associated with h are the composites fh and 1_{C}h.

4.5 Fibration versus opfibration

The use of fibrations is slightly unsatisfactory in that requirement R7B has an interpretation in the enterprise model which is similar to a conceptual model mandatory constraint, which appears at the *one* end of the relationship rather than the *many* end, as is customary in ER modelling.

Using instead requirement R7F results in opfibrations rather than fibrations. However, this approach is also slightly unsatisfactory, as can be seen from the examples in Figs. 10 and 11.

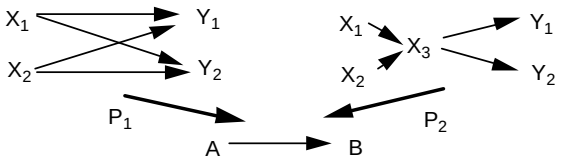


Fig. 10. Left functor not a fibration, right functor is

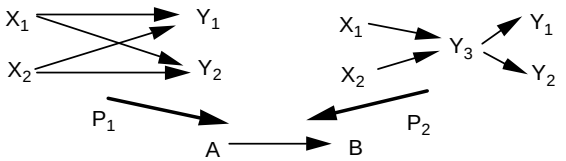


Fig. 11. Left functor not an opfibration, right functor is

Figure 10 is an application of the axioms for fibrations. The functor on the left (P_1), whose target is A$\rightarrow$B and whose source is the left category with the X_is taken to A and the Y_is taken to B, fails to be a fibration, while the similar functor on the right (P_2) satisfies the requirements for a fibration.

The analogous situation for opfibrations is shown in Fig. 11 – the functor on the left is not an opfibration, but the functor on the right is.

In Fig. 10 the schema fragment $X_3Y_1Y_2$ could be implemented as a universal relation [e.g. 13], while the schema fragments $X_1X_2Y_1$ and $X_1X_2Y_2$ cannot. In Fig. 11 none of, $X_1X_2Y_1$, $X_1X_2Y_2$, and in particular $X_1X_2Y_1$ have the universal key property. Neither in Fig. 11 is there a compelling reason from the theory of databases why any of these schema fragments should be more valid than any other.

Thus fibrations have the advantage of satisfying a useful design principle, the universal key property, while opfibrations have the advantage of being in keeping with the mandatory nature of relationships.

This analysis leads us to slightly prefer the fibration axioms over the opfibration axioms. In practice, most of the results below are consequences of the definition of a cartesian arrow. If the use of fibrations as an abstraction mechanism passes the test of useful applications, it may be possible to develop a slightly different definition which would remove the small anomaly.

4.6 R8, R9 and the concept of right inverse functor

The category-theoretic system closest to R8 and R9 is the concept of a *right inverse functor.* A (surjective) fibration is a surjective mapping from the implementation model to the enterprise model. R8 and R9 are satisfied by injective mappings from the enterprise model to the implementation model, which satisfy certain properties. The case study from which Figs. 1 and 3 come is shown to satisfy these requirements.

4.7 Discussion

We have argued so far that it is possible to specify the relationship between an enterprise model and the conceptual model of an information system implementing it in such a way that the enterprise model can be considered as an abstraction of the implementation model, and that the notation of the enterprise model can be given a precise semantics closely related to the semantics of ER modelling. Further, we have shown that the category theoretic notions of (surjective) fibration satisfies the first seven requirements for the definition of an abstraction, and have argued that the stronger concept of right inverse can be developed into a suitable precise definition of the two strongest requirements. We now present a collection of examples and propositions supporting the practical application of the fibration concept, together with an extensive case study illustrating this application. In the following there is also a similar treatment of the concept of the necessary specialised right inverse.

5 Fibrations

The definition of fibration given above is complex. We begin by disarticulating the definition in the context of testing whether a candidate functor is a fibration. To check whether P:C$\rightarrow$ E is a fibration,

F1: We have to check every arrow in E
F2: For each arrow f : C $\rightarrow$ B in E, we have to check for each object Y above B that there is a cartesian arrow c above f whose target is Y.

To check whether an arrow c: Z $\rightarrow$ Y is cartesian

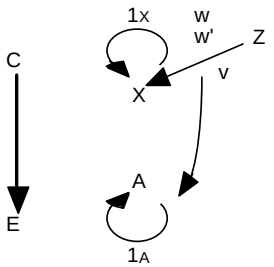
C1: We must check the composite arrow fh for every h in E with target(h) = source(f).
C2: For each fh we must check all the arrows v above fh with the same target as c. There must be at least one such arrow.
C3: For each v, there must be a unique w above h such that cw = v. We break this condition into two:

C3E: a w exists
C3U: the w is unique

We will develop some intuition about fibrations through several examples and some useful propositions. If functor P : C $\rightarrow$ E is a fibration, we will designate C as its *finer category* and E as its *coarser category*. We will call a functor whose codomain is the category **1** (one object only with only the identity arrow) the *constant functor*.

Proposition 1 *The constant functor is a fibration*

F1: The only f in Arr(E) is 1_A.
F2: One of the arrows above f must be 1_X, for each X in Obj(C). Claim 1_X is cartesian.



C1: The only h in E is 1_A, and fh = $1_A 1_A = 1_A$.
C2: Choose any v:Z $\rightarrow$ X. PZ = A, Pv = 1_A. There is at least one, v = 1_X.

C3E: $c = 1_X$, so $cw = v \Rightarrow w = v$. ($1_X v = v$)
C3U: $1_X w' = v \Rightarrow w' = v$ by the definition of the identity.

Proposition 1 shows why fibrations give abstractions. If we call the subcategory of C above A in Obj(E) a *fibre* associated with A, we can see that the fibration abstracts away from any structure not involved in connections among fibres.

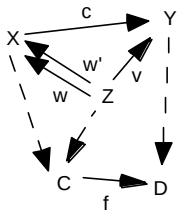
Proposition 2 *The identity functor is a fibration*

For every f in Arr(C), f is above f (F1, F2). For any w which composes with f (C1), $fw = fw$ (C2, C3E), and is unique, since $I_C w' = w \Rightarrow w' = w$ (C3U).

Proposition 2 shows that we can abstract part of an implementation model, but leave the rest unchanged.

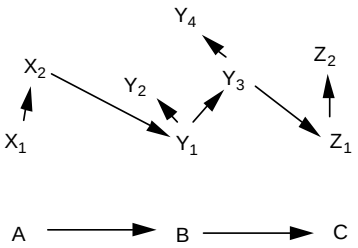
Note that in all the examples identity arrows are left out to avoid cluttering the figures.

Example 1 The following fails to be a fibration, assuming that $cw = cw' = v$, but $w \neq w'$.

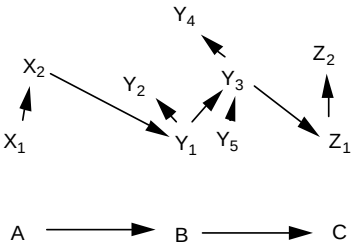


The arrows above f are c and v. C3E is satisfied by c since $P(cw) = P(cw) = f1_C$, but not C3U, since $w \neq w'$. C3E is not satisfied by v, since $Pu = f1_C$, but there is no arrow : $X \rightarrow Z$ at all.

Example 2 The following is a fibration ($\{X_i\}$ above A, $\{Y_i\}$ above B, $\{Z_i\}$ above C):

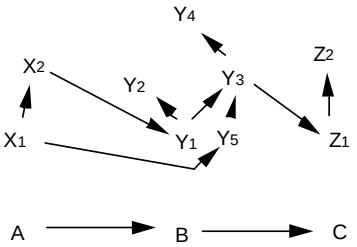


Example 3 The following is not a fibration

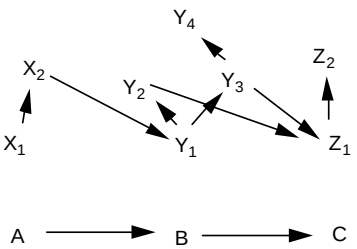


because there is no arrow above A→B whose target is Y_5 (F2).

Example 4 The following corrects the problem with Example 3:

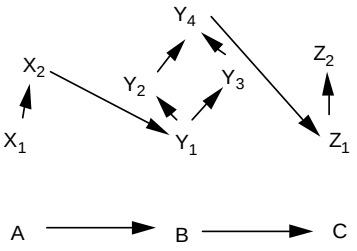


Example 5 The following is not a fibration



because both arrows $Y_2 \to Z_1$ and $Y_3 \to Z_1$ fail C3E.

Example 6 The problem in Example 5 can be corrected by the following:



As a technical aside, we can call the set of arrows of C above some f in Arr(E) with target X above target(f) the objects of a *slice category* with centre X, which we can name slice(f, X). If we denote by fibre(A) the fibre above A in Obj(E), the arrows of slice(f, X) are the arrows of fibre(source(f)) with the property that if h, h' in Obj(slice(f, X)), then g in Arr(fibre(source(f))) is in Arr(slice(f, X)) if h = h' g. These examples show that a cartesian arrow c for f and X is the terminal object of slice(f, X). In Example 6, the objects of slice(B→C, Z_1) are $\{Y_1 \rightarrow Z_1, Y_2 \rightarrow Z_1, Y_3 \rightarrow Z_1, Y_4 \rightarrow Z_1\}$, the arrows are $\{Y_1 \rightarrow Y_2, Y_1 \rightarrow Y_3, Y_1 \rightarrow Y_4, Y_2 \rightarrow Y_4, Y_3 \rightarrow Y_4\}$, and the cartesian arrow $Y_4 \rightarrow Z_1$ is the terminal object. This gives us a simple way to recognise whether a functor is not a fibration, if it has structures like that of Example 5. See Proposition 5 below.

Criteria F1 and C1 require that we check every arrow in the target of the functor. It is convenient to consider three classes of such arrows:

A1: the identity arrows
A2: the non-identity arrows, where C1 is satisfied by the identity on the source.
A3: the non-identity arrows, where C1 is satisfied by a non-identity arrow.

Proposition 3 *Checking A1 can never lead to a functor failing to be a fibration*

Let 1_A be an identity in **E**. For every X in **C** such that PX = A, claim that 1_X is a cartesian arrow for $1_A$and X (F2). The only arrow in E satisfying C1 is 1_A. Let v in Arr(**C**) be such that $Pv = h1_A$ (C2). Then $v1_X = v$ (C3E), and $v1_X = v'1_X \Rightarrow v = v'$ (C3U).

The following propositions assist in checking A2. We need some definitions. If f : C → D is in E and Y is above D, then denote by C(Y) the subcategory above C whose objects are the source of arrows whose target is Y. Two distinct arrows with the same source and target are said to be *parallel*.

Proposition 4 *If A2 is satisfied, C(Y) is not empty for all Y above D.*

This is a restatement of F2.

An object X of a category C is *terminal* if for every Y in Obj(C), ∃!f:Y→X. In other words, every object in the category is the source of a unique arrow whose target is X.

Proposition 5 *If p : X → Y is a cartesian arrow above f , then X is a terminal object of C(Y).*

Assume otherwise. If X is not a terminal object of C(Y), there is an X′ above C and an arrow v : X′ → Y such that either:

i. There is no arrow q : X′ → X. In this case C3E fails for v. (See Example 5.)
ii. There are two distinct arrows q, q′ : X′ → X. Both q and q′ are above 1_C and in this case C3U fails for v. (See Example 1.)

An *endomorphism* is an arrow whose source and target are the same object. An identity arrow is an endomorphism.

Proposition 6 *If there are two parallel arrows p, p′: X → Y and p is a cartesian arrow, then C(Y) has a non-identity endomorphism.*

If p is cartesian, then v = p′ is above $f1_C$. The unique w such that pw = p′ has source and target X, so is an endomorphism. The arrows p and p′ are distinct, so w ≠ 1_X.

Proposition 7 *If (i) C(Y) has a terminal object X, and (ii) there are no parallel arrows from X′ to Y where X′ in C(Y), then p : X → Y satisfies the condition A2.*

F2 is satisfied, since if C(Y) has a terminal object, there is at least one arrow p above f. Every arrow v above $f1_C$whose target is Y has its source an object X′ above C. X′ is in C(Y), so by definition of a terminal object, ∃!w : X′ → X. If pw = v′ ≠ v, then v and v′ are parallel arrows, contradicting assumption (ii).

Corollary 8 *If P is the identity functor on subcategory C′ of C, then every arrow in C′ satisfies A2.*

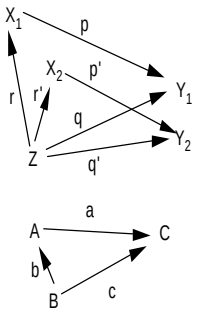
Let f : B → D be an arrow in C'. B is the only object above B, and f : B → D. P(D) has a terminal object and there is only one arrow, so Proposition 7 applies.

An *initial object* is the dual of a terminal object. An object X of category C is initial if for every Y in Obj(C) ∃!f:X→Y. In other words, every object in the category is the target of a unique arrow whose source is X.

Corollary 9 *If the arrows above f are derived from an arrow p whose source is a terminal object of the subcategory above C and whose target is an initial object of the subcategory above D, A2 is satisfied.*

We have so far established that to check whether P is a fibration, we need not check the identity arrows of E (by Proposition 3) and also that we need not check any arrows on which P is the identity (by Corollary 8). Propositions 4, 5 and 6 give some useful tests by which to fail a candidate for a fibration, and Proposition 7 and Corollary 9 give tests for satisfaction of A2 in useful special cases. After some examples involving A3, Proposition 10 below gives a test for A3 in a special case which includes the conditions for Proposition 7.

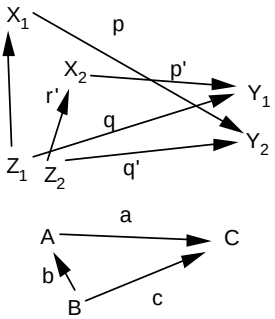
Example 7 Commuting diagram in a fibration



p is a cartesian arrow for a and Y_1, while p′ is a cartesian arrow for a and Y_2.

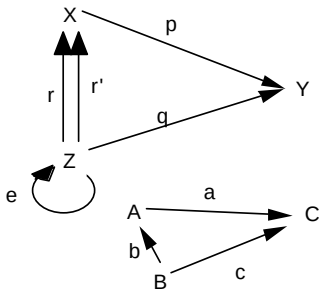
Examples 8 and 9 show that satisfaction of A2 for arrows in a commuting diagram is not enough for the diagram to satisfy A3.

Example 8



Try u = p, then v = q′, which fails C3E. Try u = p′, then v = q, which also fails C3E.

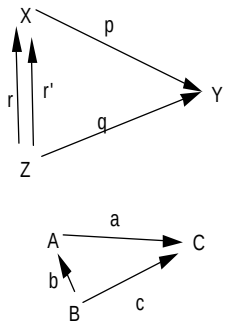
Example 9



If r ≠ r′, but r = r′e and pr = pr′, then r above b is cartesian. v = r, w = 1_Z; v = r′, w = e. However, p is not cartesian for a and Y, since both r and r′ are above b (fails C3U).

Proposition 10 *If each arrow of a commuting diagram in E passes A2, and there are no non-identity endomorphisms in C, then if a commuting diagram in C passes C3E it also passes C3U.*

Assume the contrary. For u to satisfy C3E but not C3U, the following diagram must obtain.



Consider the diagram above b. Try r as a cartesian arrow. v = r succeeds with h = 1_Z, but v = r′ fails, as there is no w. r′ fails by symmetry to be cartesian.

We complete this section by noting that a composition of fibrations is a fibration, so it is possible to proceed in stages.

6 Application to extended example

Any mathematical concept is useful only if it gives suitable results in practice. In this section, we apply the machinery of the previous section to two extended examples based on Figs. 1 and 2.

6.1 Enterprise model

Figure 12 is a representation of Fig. 1 as a category. Although its semantics are not precise at this stage, we give an intuitive idea of the intended interpretation of this enterprise model, which will help in making sense of it as an abstraction of a concrete information system specification. A *Product* is something the corporation sells. A *Resource* is usually the result of a (capital) investment of some sort. It is something that can be consumed, including usage of an asset for a period of time. *Capability* is an association between *Product and Resource* representing a static relationship, while *Activity* is a dynamic association between *Product* and *Resource*. A *Transaction* is a reason for performing an *Activity* with respect to a *Product*, which is done in respect of an *Agreement*. *Performance Conditions* are aspects of an agreement with respect to *Products*. *Transactions* are associated with *Performance Conditions* via an association called *Link*.

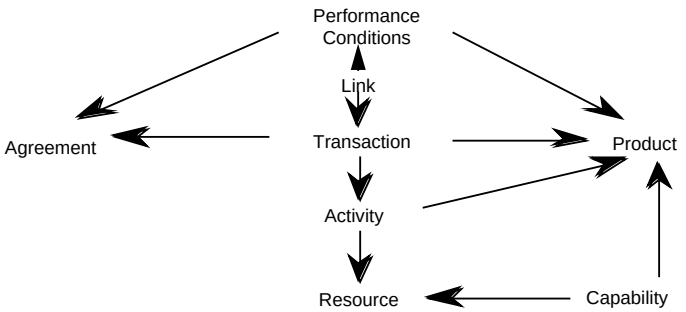


Fig. 12. Figure 1 represented as a category

In Fig. 13 we present a detailed ER diagram representing part of a proposed information system for the same organization (Fig. 3 with three named shaded areas). We take the position that the two models, although developed for the same organization, have been developed independently. Our task is to assume that the abstract model is intended to be a fibration abstraction of the detailed model, and to check that this is so. Since the constraints involved in a fibration are very strong, we would not expect that the test would immediately succeed. In practice, the discrepancies could either be

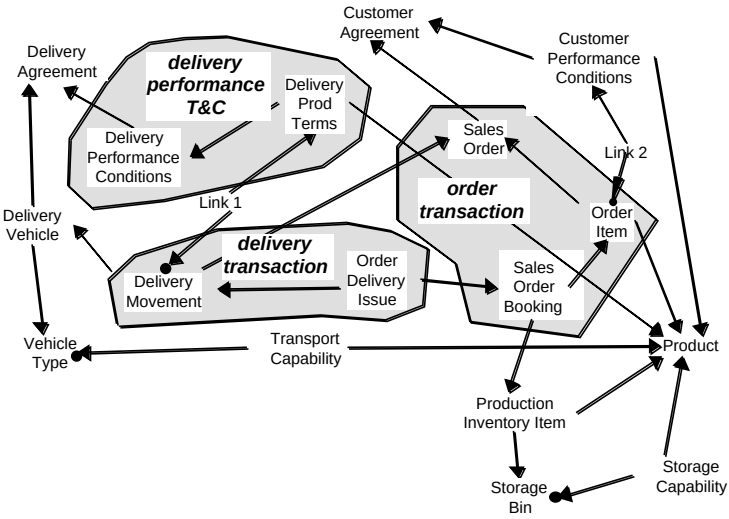


Fig. 13. An information system implementation model for the organization of Fig. 12

errors in the information system or the abstract model, but for this exercise we will assume that the abstract model is correct. When a discrepancy is found, therefore, we will attempt to propose plausible modifications to the information system which allow the abstraction to be a fibration.

Note that some of the arrows are designated as epi (dot on the arrowhead). The category-theoretic concept epi is a generalisation of a surjective mapping. There is a parallel concept *monic*, which is a generalisation of an injective mapping.

Since testing for a fibration is a complex process and fibration is preserved over composition, we will begin with a preliminary abstraction. This has three advantages: first, it illustrates the basic principles on a small scale; second, we can proceed with a smaller implementation model if we work with the preliminary abstraction; and third, working with already abstract entities will illustrate some issues in the process.

There are three enterprise entities shown in Fig. 13 as shaded areas: *Delivery Performance T&C*, which is a generalization of *Delivery Performance Conditions* and *Delivery Prod Terms*; *Delivery Transaction*, which is a generalization of *Delivery Movement* and *Order Delivery Issue*; and *Order Transaction*, which is a generalization of *Sales Order, Order Item, and Sales Order Booking*. These are intended to be intuitively plausible generalizations, respectively contract terms, entities associated with deliveries, and entities associated with orders. We assume that all diagrams commute.

To check that this is a fibration, by Corollary 8 we need only to check the coarser arrows whose source or target are in the shaded areas. The arrow: *Delivery Performance T&C* $\rightarrow$ *Delivery Agreement* passes A2 by Corol-

lary 9, as does the arrow : *Link 1* → *Delivery Performance T&C*. The arrow : *Delivery Performance T&C* → *Product* passes A2 by Proposition 7, as does the arrow : *Delivery Transaction* → *Delivery Vehicle*. However, the arrow : *Link 1* → *Delivery Transaction* fails A2 by Proposition 4.

Link 1 is an association between *Delivery Movement* and *Delivery Product Terms*. The latter is associated with a specific *Product*, while the former is associated with a specific *Sales Order*, which is associated with many *Products* via its association with many *Order Items*. On the other hand, *Order Delivery Issue* is associated with a single product and a single *Delivery Movement.* It is plausible that the arrow whose source is *Link 1* should have its target *Order Delivery Issue* rather than *Delivery Movement*, because then both parts of the association of *Link 1* are associated with a specific product. We show this correction in Fig. 14.

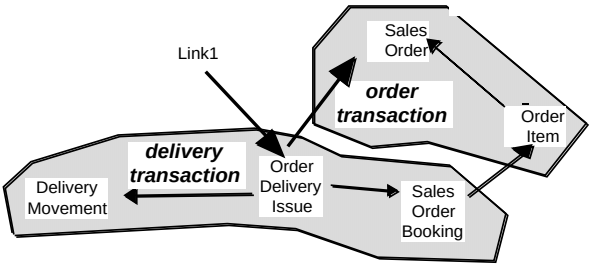


Fig. 14. Modifications to Fig. 13 so that the abstraction is a fibration

The final coarser arrow from *Delivery Transaction* has target *Order Transaction*. The finer arrows above it are *Delivery Movement* → *Sales Order* and *Order Delivery Issue* → *Sales Order Booking.* Both arrows pass A2 by Proposition 7 since the diagram commutes. The arrow *Order Transaction* → *Customer Agreement* satisfies A2 by Corollary 9. The arrow *Order Transaction* → *Product* passes by Proposition 7, as does the arrow *Order Transaction* → *Production Inventory Item.* This leaves only the arrow *Link 2* → *Order Transaction*, which fails A2 by Proposition 4.

This problem with *Link 2* cannot be corrected in the same way as the problem with *Link 1*. A *Sales Order Booking* is essentially an arrangement for possibly partial delivery of an *Order Item*, which occurs after the *Order Item* is in place. On the other hand, it is plausible to regard *Sales Order Booking* as part of the *Delivery Transaction* rather than the *Order Transaction.* If this change is made in the coarser category, the problem with *Link 2* is corrected by Corollary 9. A new problem is created, however. The coarser arrow *Delivery Transaction* → *Order Transaction* now fails A2 by Proposition 5 (there is a derived arrow *Sales Order Booking* → *Sales Order*).

The entity *Delivery Movement* is an association between *Order Delivery Issue* and *Delivery Vehicle*, reflecting the assignment of a delivery vehicle to a (possibly partial delivery) of an order item. The arrow *Delivery Movement* $\rightarrow$ *Sales Order* appears to be a consistency check to make sure that the delivery is in respect of the correct order. It is plausible that the check should be made at the time of creation of an instance of *Order Delivery Issue*, rather than at the time of the assignment of a vehicle to the delivery. The plausibility of this correction is increased if we notice that this change corresponds to the change made in respect of *Link 1*. If this change is made, the coarser arrow passes A2 by Proposition 7. (The only cartesian arrow for *Delivery Transaction* $\rightarrow$ *Order Transaction* and *Sales Order* is the derived arrow whose source is *Sales Order Booking*.) The remaining coarser arrows whose source is *Delivery Transaction* still pass, including the new one whose target is *Production Inventory Item* (an effect of moving *Sales Order Booking*). This correction is also shown in Fig. 14 (the two bold arrows are the corrections). Inspection of the new diagram using Proposition 10 shows that A3 is satisfied, so that the abstraction is a fibration.

This preliminary task has illustrated two benefits of our method of abstraction. First, it has suggested semantically plausible improvements in the implementation model. Recall that the implementation model of Fig. 3 was constructed as a design exercise, and therefore possibly contains errors. Second, we have shown that the original choice of abstraction was not well-supported by the structure of the model; so that despite its name, the entity *Sales Order Booking* is better considered as part of the delivery-related abstraction rather than the order-related abstraction.

Figure 15 contains two diagrams. The diagram on the left is the coarser diagram resulting from the process so far. The diagram on the right is the diagram of Fig. 12. The names of the entities have been abbreviated in both cases. The diagram on the left has been re-arranged and its entities grouped so that the diagram on the right is visibly an abstraction. The finer arrows on the left are numbered (the numbers are not consecutive for historical reasons), and the coarser arrows on the right are labelled a–k. The ellipses shown on the left diagram correspond to the coarser entities on the right diagram. The arrangement is intended to show the subcategories above each of the coarser entities, and the arrows above each of the coarser arrows.

We will systematically check condition A2 for each of the coarser arrows in order from a to k. However, we first notice that the correspondence fails to be a functor, because arrows 19 and 23 are not represented in the coarser diagram.

The semantics of arrow 23 is that a delivery vehicle instance is specific to a single customer agreement instance. If this is the case, then the coarser diagram must be changed. It is plausible, however, that although a vehicle

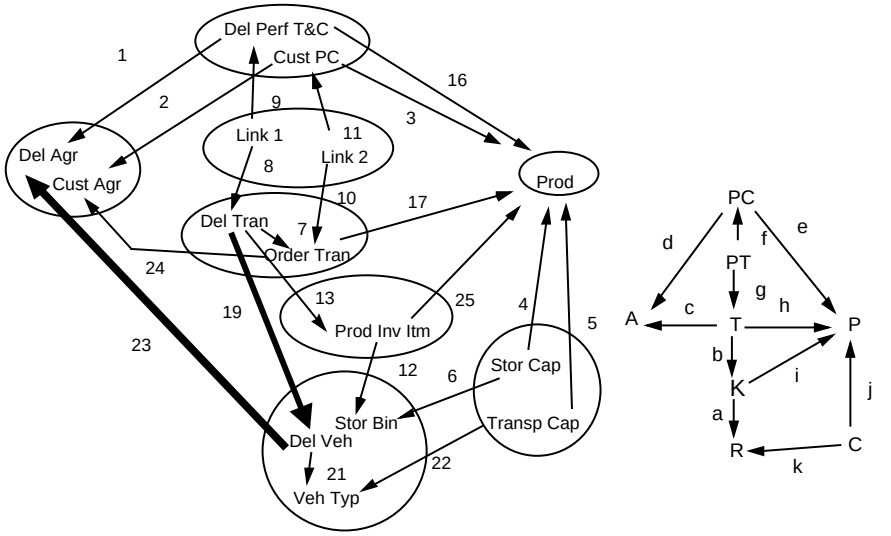


Fig. 15. Correspondence between the corrected abstraction of Fig. 13 and Fig. 12

is used for a particular delivery transaction (*Del Veh* is determined by *Del Tran* via arrow 19), the vehicle can be used for other delivery agreements. This suggests that the finer diagram is in error, and arrow 23 should be made a direct arrow from *Del Tran* to *Del Agr*.

Arrow 19 is a coarser arrow. It is derived from the finer arrow of Fig. 13 *Delivery Movement* $\rightarrow$ *Delivery Vehicle*. Notice that the absence of *Delivery Movement* from the coarser entity *Delivery Transaction* does not affect any of the other three arrows involving this coarser entity. We propose therefore to leave *Delivery Movement* out of the *Delivery Transaction* abstraction, and include it as an *Activity* along with *Production Inventory Item*. This involves replacing arrow 19 with arrows 26 (derived ultimately from the arrow *Order Delivery Issue* $\rightarrow$ *Delivery Movement*) and 27. The alterations to Fig. 15 are shown in Fig. 16. The alterations to Fig. 13 are cumulated with subsequent changes in Fig. 23 below.

We have another case of an intuitively appealing abstraction being structurally inappropriate. Despite its name, the entity *Delivery Movement* is better generalized as an *Activity* rather than as a *Transaction*. This structural error was not discovered until the first generalization was used as a component of a larger generalization.

We are now in a position to systematically apply the tests for A2. Coarser arrows a, b, c and d pass A2 by Proposition 7. Coarser arrow e fails A2 by Proposition 5.

There is no obvious reason why one should expect there to be a functional association between *Del Perf T&C* and *Cust PC*. However, we notice that

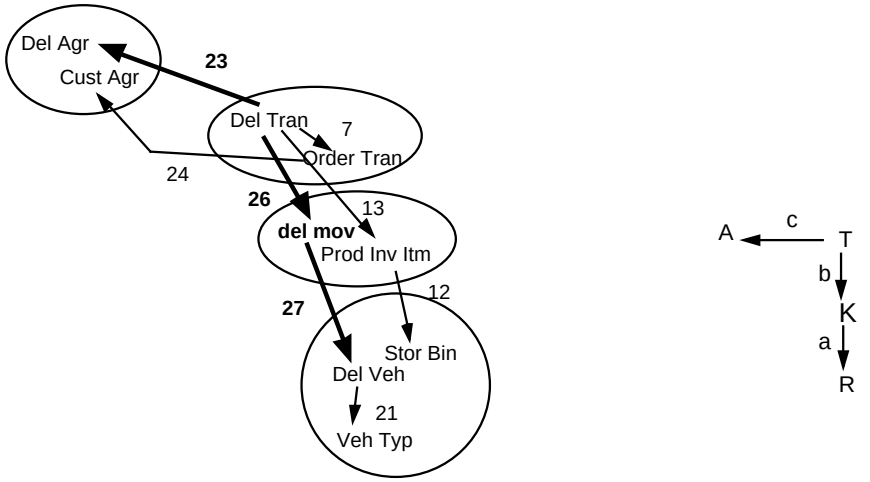


Fig. 16. Alterations to Fig. 15 so that the correspondence is a functor

both entities depend on a single product. It is plausible that each instance of *Del Perf T&C* should correspond to an instance of *Prod* for which an instance of *Cust PC* exists. In this case, we can identify the subset of *Prod* for which both exist, and require that subset entity to be the target of arrows 3 and 16. These two arrows are now internal to the coarser entity *PC*, and the external arrow 28 is the subset inclusion. These changes are shown in Fig. 17. (Categorically, we have factored both arrows 3 and 16 into an epi composed with a monic, and have identified the two intermediate objects. Semantically, the change is the addition of an integrity constraint.)

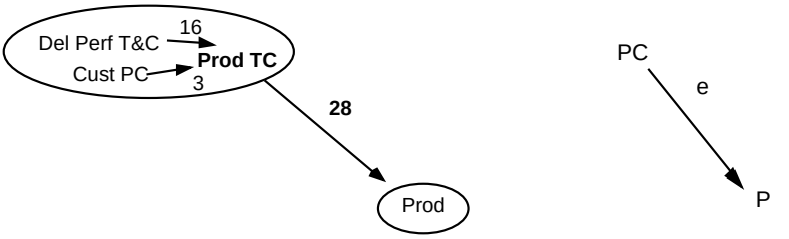


Fig. 17. Alteration to Fig. 13 correcting a problem with coarser arrow e

Both coarser arrows f and g fail A2 by Proposition 5. Since coarser entity PT is an abstraction of two association entities, it makes sense to consider the two arrows together. This problem is somewhat subtle. From a Category Theory point of view, the problem is that the subcategories above PC and T both have objects which are the target of arrows from both objects above PT, and the subcategory above PT does not have a terminal object

(Proposition 7). Furthermore, T is a coarser entity. Any changes must be made to the finer model of Fig. 13 as amended. Figure 18 shows the relevant fragment of Fig. 13 with the modifications introduced for Figs. 14 and 17.

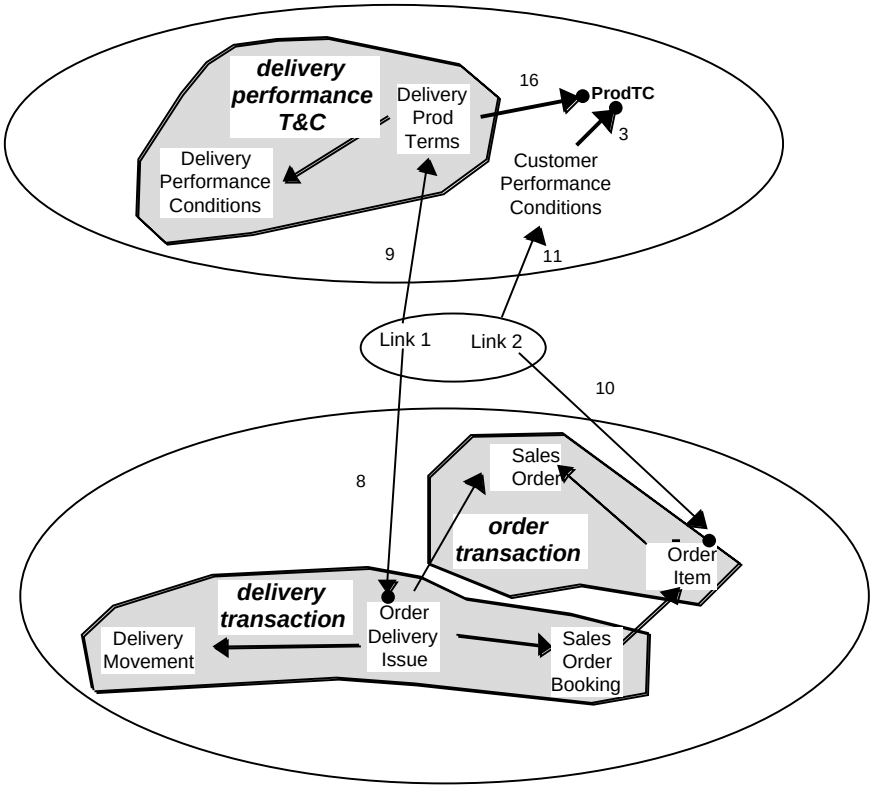


Fig. 18. Fragment of Fig. 13, with modifications from Figs. 14 and 17.

Structurally, the problem can be solved with an arrow *Link 1* $\rightarrow$ *Link 2*. This arrow is in fact implicit in the diagram. Arrow 10 is epi. In this case, we can assume that there exists in the enclosing category at least one right inverse (this is true of FinSet). The tactic is to select one of the right inverses for inclusion in the diagram. This generates an indirect arrow : *Link 1* $\rightarrow$ *Link 2*. If we require that the diagram created by the new arrow involving *Product TC* commutes, both coarser arrows f and g now pass A2 by Proposition 7.

This modification introduces another problem. Now there is an coarser arrow T $\rightarrow$ PT. This new arrow fails A2 by Proposition 4. This new problem can be corrected by recognizing that arrow 8 is also epi, and by introducing one of its right inverses. Figure 19 shows the requisite modifications to Fig. 15, with the introduction of arrow 29. The arrows deriving from the

epis are not shown directly, but indicated by showing both arrows 8 and 10 as epi as well as coarser arrow g.

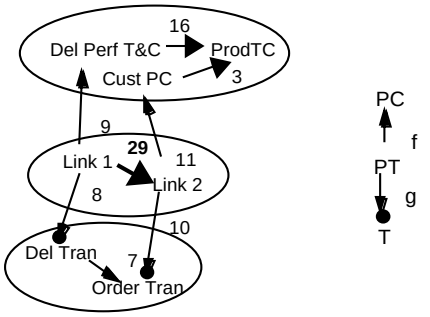


Fig. 19. Corrections to Fig. 15 for coarser arrows f and g

Semantically, that arrows 10 and 8 are epi represents the constraint that each *Order Item* in Fig. 13 must be associated with at least one *Customer Performance Condition*, and that each *Order Delivery Issue* must be associated with at least one *Delivery Product Term*. The right inverse arrows say that given an Order Item, one can find a *Customer Performance Condition*, and that given an *Order Delivery Issue*, one can find a *Delivery Product Term*. From the category theory point of view, only one of the possibly many right inverses in each case can be selected for inclusion in the diagram, otherwise parallel arrows are introduced and the coarser arrows fail A2 by Proposition 6.

We have assumed that all diagrams in the implementation model of Fig. 13 commute, and apply that assumption to our subsequent modifications up to date. We need to be able to choose the right inverses so that the new arrow 29 commutes as required above. The situation is diagrammed in Fig. 20, derived from Figs. 18 and 19.

We want to establish whether we can choose arrows i and i′ such that the diagram $\{[16, 9], [3, 11, 29]\}$ commutes. Recall that $29 = [i', p, p', 8]$. We have by assumption

$$[28, 3, 11] = [25, 10]$$

so

$$[28, 3, 11, i'] = [25, 10, i'] = 25$$

since i′ is a right inverse of 10. We therefore have

$$[28, 3, 11, i', p, p', 8] = [25, p, p', 8] = [28, 16, 9]$$

by assumption. Arrow 28 is monic by construction, so

$$[3, 11, i', p, p', 8] = [16, 9]$$

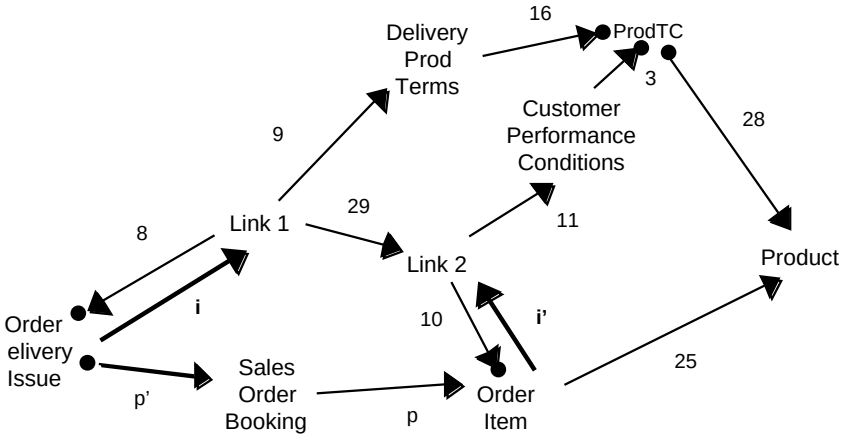


Fig. 20. Diagram needed for analysis of arrow 29

or

$$[3, 11, 29] = [16, 9]$$

as required. This shows that

i. The assumption that the arrows above the diagram in Fig. 18 commute is enough that arrow 29 has the right properties for any choice of i′;
ii. The choice of i is arbitrary with respect to the properties of 29.

We emphasize that the exercise involving arrows f and g has required no additional constraints on the application, and has introduced no additional entities or relationships. The correction has been made by a purely technical process. A CASE tool supporting fibration abstraction would have a library of such structures, and be able to automatically recognize common situations of this kind.

Having completed arrows f and g, we can move on. Coarser arrow h passes A2 by Corollary 9, and coarser arrow i passes A2 by Proposition 7. Coarser arrow j fails A2 by Proposition 5, in the same way as coarser arrow e, and admits a similar solution, viz the creation of a subobject PC of *Prod* above C, and require that the epi/monic factorization of arrows 4 and 5 both pass through PC. This has the plausible semantics that any product which has a storage capability also has a transport capability, and vice versa.

Finally, coarser arrow k fails A2 by Proposition 4, since *Del Veh* is not the target of any arrow above arrow k. A plausible additional constraint is that every vehicle type has at least one vehicle associated with it, which makes arrow 21 epi, so that we have right inverses. We can choose one arbitrarily. The situation at the end of the test A2 is shown in Fig. 21.

We now need to apply test A3 on the commuting diagrams of Fig. 21. Commuting diagram {[d, f], [c, g]} passes by Proposition 10 since [1, 9] =

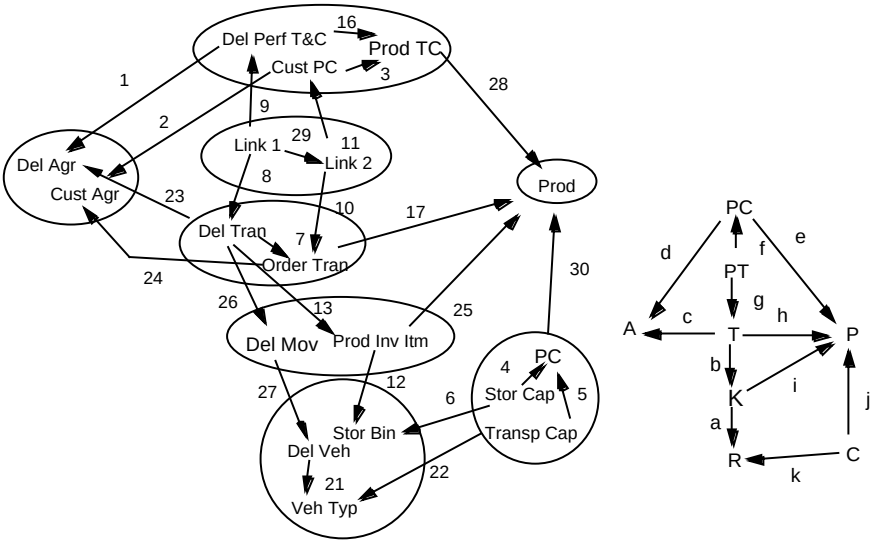


Fig. 21. Figure 15 showing the corrections for A2

[23, 8] and [2, 11] = [24, 10]. Commuting diagram {[e, f], [[h, g]} passes by Proposition 10 since [28, 2, 11] = [17, 10]. Finally, {h, [i, b]} fails A3, since the only choice for c is 25, and C3E fails for v = 17. The situation would be corrected if there were an arrow *Order Tran* → *Prod Inv Itm*, since this would satisfy C3E for v = 17. Coarser arrow i would pass A2 in the proposed situation by Corollary 9.

However, *Prod Inv Itm* is a specific choice of bin for the product. At the time of order, the specific bin may not have been assigned yet. We can achieve the desired result by adding the integrity constraint that a product ordered must have at least one storage bin assigned to it. This constraint can be implemented by adding IVP above K, and arrows 31 and 32. IVP are the products for which storage bins have actually been assigned, and arrow 32 expresses the constraint that a product ordered must be a product for which at least one bin has been assigned, as required.The diagrams shown in Fig. 22 therefore represent a fibration. Note that arrow 17 is now derived, and as a consequence, so is coarser arrow h.

We have been able to achieve this result by a combination of semantically plausible modifications to the original E/R diagram of Fig. 13, shown in Fig. 23, and incorporation into the model of selected right inverses for epi arrows. The entities of Fig. 13 have been rearranged and grouped to show the fibration.

We conclude so far that the fibration mechanism has provided a successful method of assigning a semantics to an enterprise model, and of identifying the necessary constraints in the overlying information system

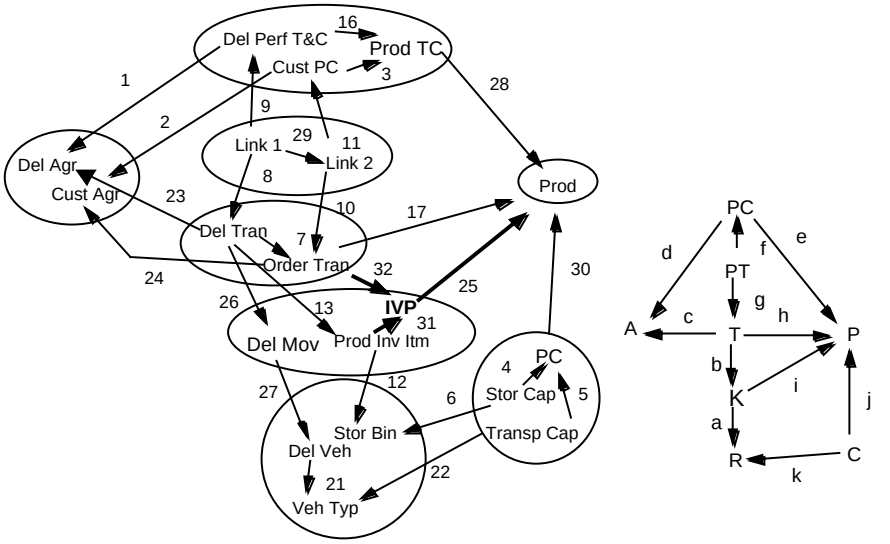


Fig. 22. Modification of Fig. 21, showing a fibration

that the enterprise model can be a template for the information system in a precise and testable way, at least for this example. It is therefore a method which deserves further investigation.

6.2 Abstract schema

The second class of problem mentioned in the introduction for which an overview of a conceptual model is required is the abstract schema, an example of which is shown in Fig. 2. We have performed a second experiment, applying the fibration mechanism to that example. Fig. 24 shows on the right the abstract schema of Fig. 2 expressed in category theory notation, and on the left the concrete conceptual model of which it is meant to be an abstraction. In both cases, some of the relationships have been identified as many-to-one, where that seemed plausible.

Arrow 14 expresses that an employee is born in a city, while arrow 12 expresses that a company is located in a state. Arrow 17 expresses that an employee works for a department, while arrows 15 and 16 express the history of departments for which an employee has worked in the past.

Coarser arrow a satisfies A2 by Proposition 7, as does coarser arrow b. Coarser arrow c fails A2 by Proposition 4, but this can be corrected if arrow 13 is made epi (every state has at least one associated city). A right inverse therefore can be found- for each state say its capital or the most important company location.

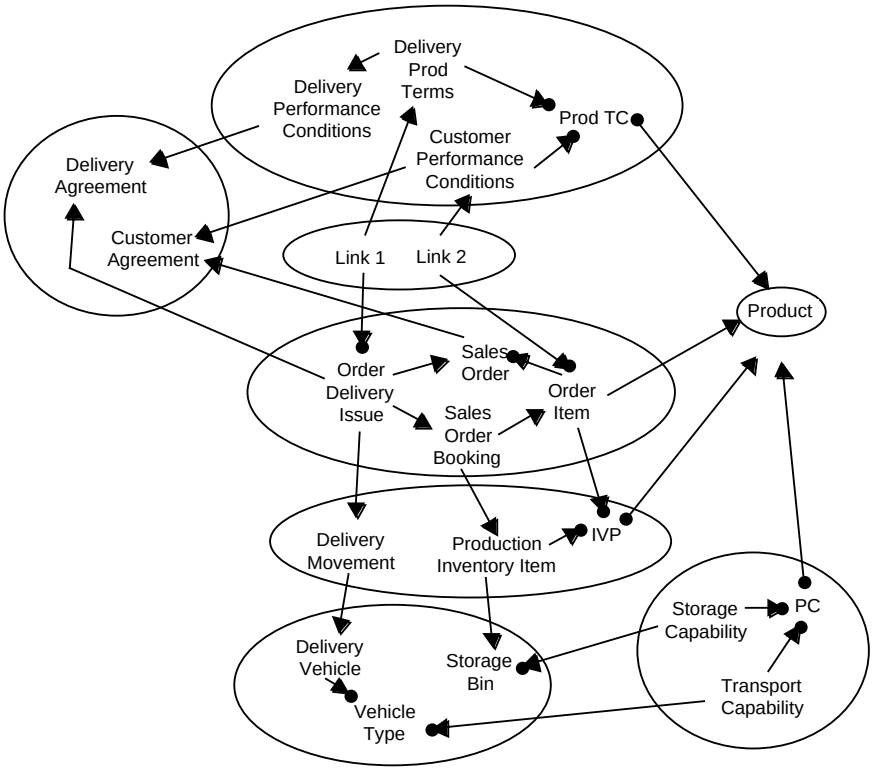


Fig. 23. Accumulated modifications to Fig. 13 allowing Fig. 22 to show a fibration

Coarser arrow d is more subtle, since the diagram {[12, 6, 17], [13, 14]} does not commute (there is no necessary relation between an employee′s state of birth and the state in which his/her company is located). There are therefore two parallel arrows above d, viz. [13, 14] and [12, 6, 17]. Since there are no non-identity endomorphisms above *Project*, test A2 fails by Proposition 6.

The problem is mainly in the coarser diagram. If as would seem reasonable the diagram {cb, d} does not commute, then there is an additional coarser arrow $d' = cb$ with arrow 14 above d and arrow [12, 6] above d'. In this case both coarser arrows d and d' pass A2 by Proposition 7 (in the case of d' noting that arrow 13 is epi, and we have a right inverse from the treatment of coarser arrow c). The only commuting diagram in the coarser diagram is $d' = cb$, which passes A3 by Proposition 10. The modified diagrams in Fig. 25 are therefore a fibration.

A second successful case study increases our confidence that the method of abstraction by fibrations has merit and justifies further study.

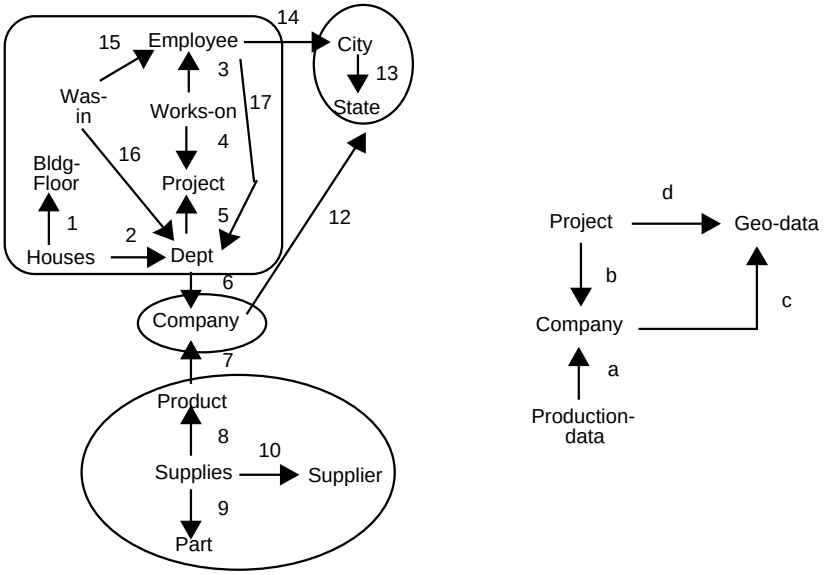


Fig. 24. An abstraction of a schema (Source: [3], p. 175)

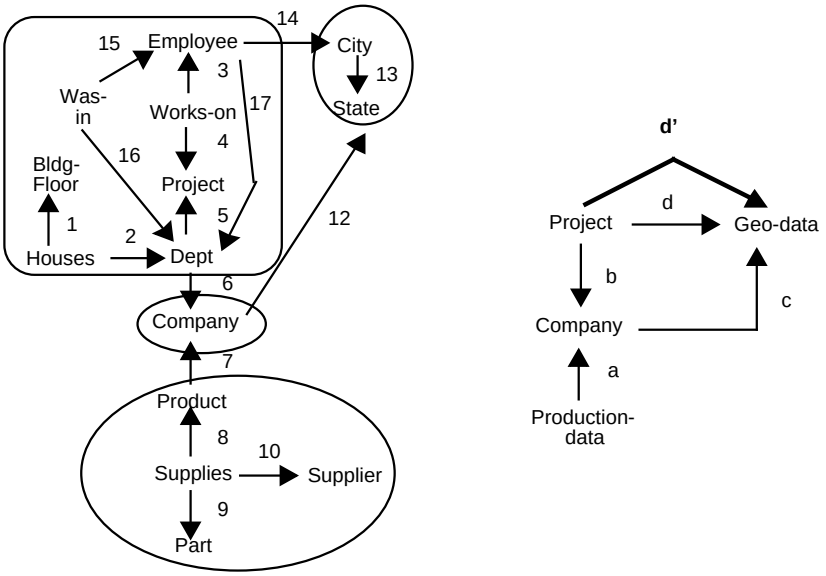


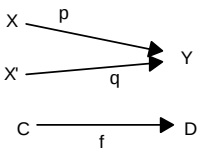
Fig. 25. Modification of Fig. 24 resulting in a fibration

6.3 Summary of tactics

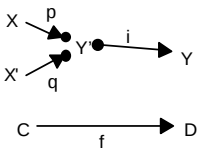
This section summarizes the main tactics used in the case study to introduce derived arrows and objects from the supercategory which has the category-theoretic structures of FinSet in order to achieve a fibration. We first need

some terminology. For testing A2, we have a proposed coarser arrow mapping a coarser source to a coarser target. We have the subcategory above each of the coarser source and target, and the arrow(s) above the coarser arrow. For testing A3, we have a proposed coarser commuting diagram, and commuting diagrams above it. A *potentially commuting diagram above the coarser commuting diagram* is a situation with c and v which fails C3E.

Inclusion of a subtype. There were cases (coarser arrows e and j) where a proposed coarser arrow fails A2 by Proposition 5, and the subcategory above the target was a single object with its identity. In these cases, the semantics of the arrows above the coarser arrow were that the object above the target was the target of two relationships whose source was above the coarser source. A plausible semantic constraint was that if an instance of the target entity was depended upon by an instance of one of the source entities, it must be depended upon by the other. The semantics call for a subtype of the target entity consisting of those instances satisfying the constraint.



becomes



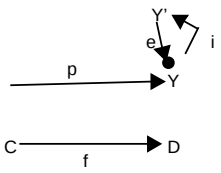
Categorically, this constraint consists of introducing the epi/monic factorization of the two arrows above the coarser arrow, implementing the constraint by identifying the two intermediate objects, and including the intermediate object in the subcategory above the source, which becomes the terminal object called for by Proposition 7.

Taking advantage of epi arrows. An epi arrow has in the supercategory at least one right inverse.

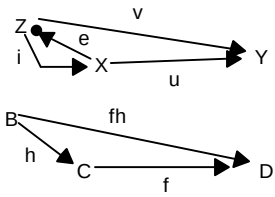
There were situations (coarser arrows f, g, k) where a proposed abstraction failed A2 by Proposition 4. There were therefore at least two objects above the coarser target. One of these objects was the target of no arrow whose source was above the coarser source, but was the source of an epi

arrow whose target was also above the coarser target, and which target satisfied Proposition 5. In these situations, it made semantic sense to include in the implementation model one of the right inverses, which satisfied Proposition 4.

For example, e is the epi and i is one of its right inverses, so there is now a composite arrow ip : X → Y′.



In testing A3, (coarser arrows b, h, i) there was a situation where there was a potentially commuting diagram above a coarser commuting diagram, where the source of c was the source of an epi arrow e whose target was the source of v. Inclusion of one of the right inverses i permitted the satisfaction of C3E and therefore satisfaction of the conditions for Proposition 10.



Non-commuting diagrams. If a category has a non-commuting diagram involving three objects, then there is always a commuting diagram with the third arrow derived from the other two, and so a pair of parallel arrows. The introduction of a fibration-based semantics to the abstract conceptual models requires the recognition of this fact. The diagrams above the commuting coarser diagram are the commuting diagrams, and those above the non-commuting coarser diagrams are the non-commuting diagrams.

7 Inverse of the fibration

7.1 The concepts

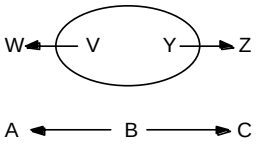
Up until now, we have been concentrating on requirements R1–R7, which we might think of as local requirements – long range perhaps, but not covering the entire model at one time.

To say that we want the enterprise model to specify large-scale constraints on the implementation model, we have proposed two requirements. First, we might want to say that the enterprise model is a template for the actual relationships existing in the implementation model. That is, if there is a connected structure in the enterprise model, then there ought to be entities above the enterprise model entities, and relationships above the enterprise model relationships, which have the same pattern. In other words, we ought to be able to select one entity above each enterprise model entity in such a way that these entities participate in the same pattern of relationships as the enterprise model. This requirement was called **R8**. A somewhat stronger requirement was **R9**: that every entity in the implementation model participates appropriately in such a pattern.

Since the category-theoretic concept of fibration was successful in giving a precise definition for requirements R1–R7, it is natural to further develop it for the last requirements. To satisfy R1–R7, we have required our fibration to be a functor which is surjective mapping of the objects and arrows of the finer category onto the objects and arrows of the coarser category. Therefore as a mapping, the fibration P has a possibly large number of right inverses, which of course are only injective mappings from the objects and arrows of the coarser category to the objects and arrows of the finer.

Further, if $P : \mathsf{C} \to \mathsf{E}$ is a fibration, then we can choose a right inverse R such that if $f : A \to B$ is in Arr(E), then R(f) is cartesian for f and R(B), which we will call a *cartesian right inverse*.

Example 11 Not every fibration surjective on objects and arrows satisifes R8. Consider



The fibration axioms do not constrain the sources of arrows, so there is no object above B which is the source of arrows with targets both W and Z.

Satisfaction of R8 requires an additional constraint on the cartesian right inverse R. The constraint required is that the cartesian right inverse mapping be a functor. The functor axioms require that the mapping preserve composition of arrows, so that the image in the finer category of the coarser category has the same structure. Diverging arrows such as in the coarser (bottom) category of Example 11 are held together by the requirement that both $B \to A$ and $B \to C$ compose with the identity 1_B.

If we take the category whose objects are C and E and whose arrows are functors, the constraint is that the functor P be epi in this category, with R a right inverse having the additional property of being cartesian. So R8 is satisfied if P has a cartesian right inverse functor.

Applying in Fig. 26 the interpretation of R8, that the fibration must have a cartesian right inverse functor, to the example of Fig. 23, we see that there is such a functor. The objects in bold are the image of the enterprise objects, and the arrows in bold are the images of the enterprise arrows. The arrows are annotated with the names of the enterprise arrows. Composite arrows are shown by dashed lines. Note that several of the composite arrows (a, b, d, h and k) depend on the existence of right inverses of epi arrows.

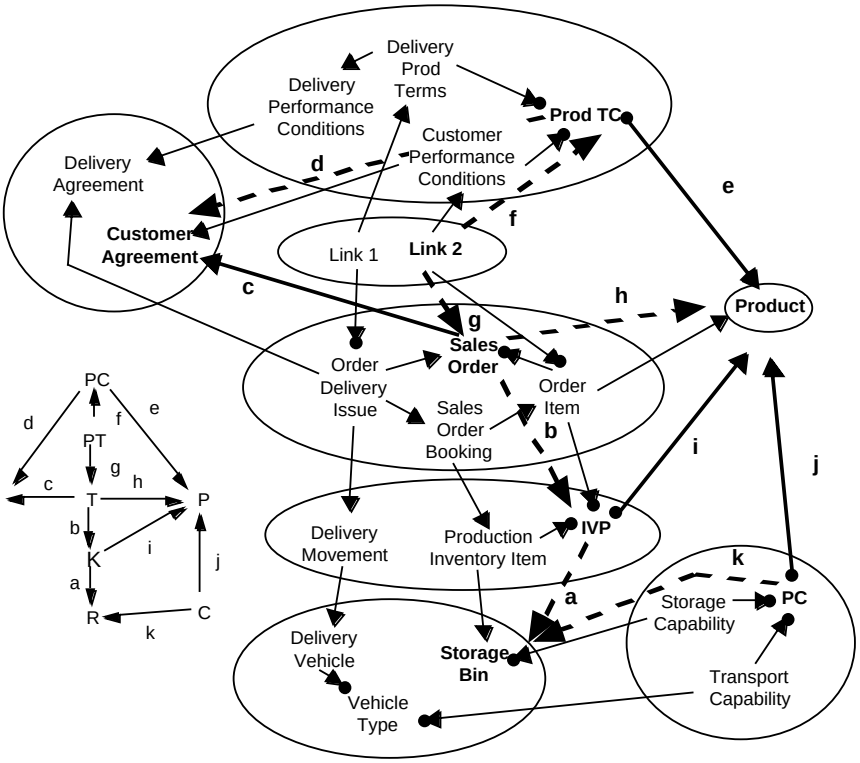


Fig. 26. The case study of Fig. 23 has a cartesian right inverse functor

7.2 R9

We have found a practicable interpretation of R8. We now turn to the final requirement R9, that every object in the implementation model participate

appropriately in an R8-type pattern. One way to interpret R9 is that every object be included in the image of a cartesian right inverse functor. This interpretation is very strong, as it requires all objects to be sources of cartesian arrows, and is far too strong to be practicable.

Looking at Fig. 26, we see that the fibre over T has a terminal object (*Sales Order*), so that the other objects are semantically well-behaved since *Sales Order* is functionally dependent on all of them. This suggests a plausible representation of R9, namely that there be a functional dependency between every implementation entity and an entity in the image of the enterprise model.

Not all of the implementation entities have functional dependencies to the image of the enterprise model shown in Fig. 26. In particular much of delivery aspect of the organisation does not. A right inverse including those entities is shown in Fig. 27. In this right inverse, there fail to be cartesian arrows above arrows e, h and i of the enterprise model.

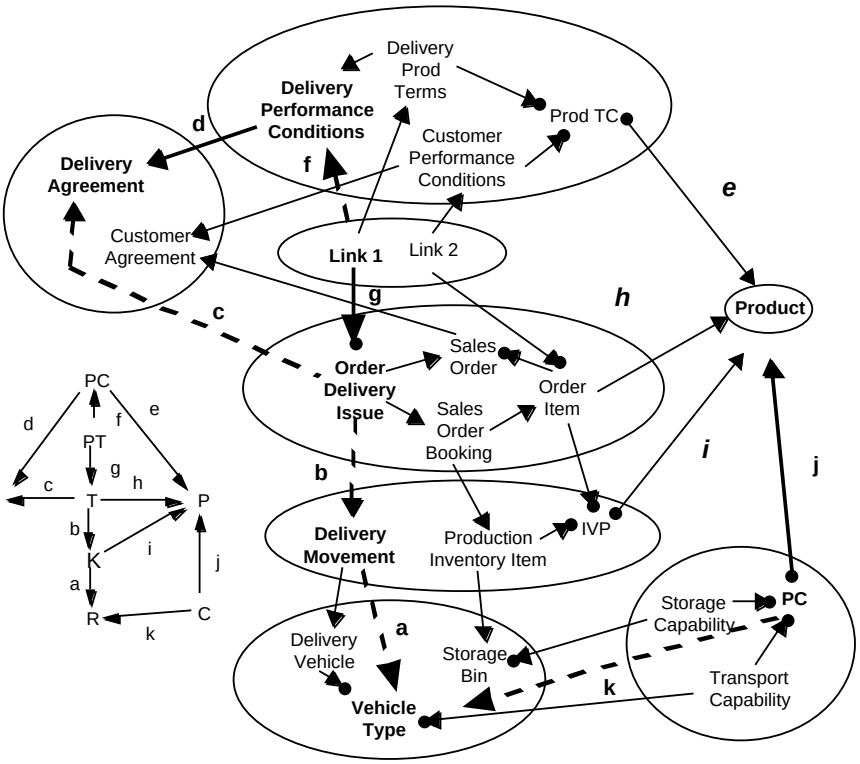


Fig. 27. A right inverse including what is missing from that in Fig. 26

All of the highlighted implementation entities in Fig. 27 have functional dependencies on entities above either A or R, that is over enterprise entities which are the target of arrows but not the source of any. Looking first at enterprise entity A, there are two implementation entities above, *Delivery Agreement* and *Customer Agreement*. It is semantically plausible that the delivery agreement should be a section of the customer agreement, so that there would be a functional dependency from *Delivery Agreement* to *Customer Agreement.* Alternatively, *Delivery Agreement* could be a set of standard agreements, but there could be an association between *Delivery Agreement* and *Customer Agreement*, which would be the source of arrows to both, and a target of the arrow from *Order Delivery Issue*. This would also satisfy R9.

Implementation entities above enterprise entity R are all resources for handling products. One way to satisfy R9 is for there to be an arrow whose source is *Delivery Movement* and whose target is *Storage Bin*. If we step back to the semantics of the enterprise model in Fig. 1, we see that an *Activity* is a mobilisation of a *Resource* in the service of a *Transaction*. The implementation model object above Activity (K) both follow this semantics. It would seem therefore that there should be a mandatory relationship between an *Activity* and a *Transaction*, in other words every arrow above c should be epi. If this were so, then *Order Delivery Issue* $\rightarrow$ *Delivery Movement* would be epi, and a right inverse would give an arrow *Delivery Movement* $\rightarrow$ *Storage Bin*, as required. Alternatively, we can see that *Vehicle Type* is as valid a choice as an image of R as is *Storage Bin*. With the previously mentioned corrections above A, the subcategory of the implementation model of Fig. 27 is also an image of a right inverse satisfying R8. Fig. 27 as amended therefore satisfies R9. (There is an arrow from *Delivery Movement* to *Product* if arrow b is epi.)

For a further feasibility test, we can apply R8 and R9 to the Batini et al. example of Fig. 25. R8 is clearly satisfied, but R9 is not. There is no arrow whose source is *Bldg-Floor* above either d or d′, and no arrow whose source is either *Supplier* or *Part* above a. The former is an omission, as a building and floor must be located in a city, giving an arrow above d′. The latter problems, above *Production-Data*, show that the model is seriously incomplete and therefore the abstract model is insufficient.

R9 is therefore a feasible integrity constraint.

8 Conclusions

This paper has argued that it is possible to specify the relationship between an enterprise model and the conceptual model of an information system implementing it in such a way that the enterprise model can be considered

as an abstraction of the implementation model, and that the notation of the enterprise model can be given a precise semantics closely related to the semantics of ER modelling. Further, this paper has discussed a number of results which indicate that the category theoretic notions of fibrations and right inverses are suitable candidates for a precise definition of this relationship.

One of the problems with abstract diagrams noted in the preceding is that it is difficult to assign a precise semantics to the coarser entities and relationships. If these coarser diagrams are related to their information systems implementation models by fibration, then an interpretation can be given.

A coarser entity represents a set of finer entities and the relationships among them. A coarser relationship represents some of the finer relationships between entities associated with its source and entities associated with its target. The set of coarser arrows between two coarser entities represent disjoint sets of relationships, and together represent all the relationships.

Furthermore, every commuting diagram in the finer model is represented in the coarser model, and there is no commuting diagram in the coarser model which does not represent at least one commuting diagram in the finer model. (Commuting diagrams are neither created nor destroyed.)

More concretely, Dampney *et al.* [4] have interpreted the dependencies represented by the arrows in the finer diagram as: if there is an arrow f : A $\rightarrow$ B, then the existence of an object of type A depends on the existence of an object of type B. This interpretation induces an interpretation of the coarser arrows which is slightly different.

The focus of the definition of fibration is on the object above the target (F2). This criterion can be interpreted as everything in B is depended upon by something in A. The criterion concerning cartesian arrows can be interpreted as that each dependency is unique. In other words, if a change is made to an entity above B which affects anything in A, it affects every entity above A which depends on the changed entity.

Procedurally, this interpretation becomes: if an instance is deleted from an entity above B, then at least one entity in A must be checked for deletion. If an instance must as a result be deleted from an entity above A, then this deletion cascades to every entity above A depending on the changed entity above B.

The entire coarser diagram can be viewed as a type system for the finer diagram. Each coarser entity is a type designator, as is each coarser arrow, and the coarser relationships determine which types of finer entities can sustain which types of relationships. The requirements for fibration impose integrity constraints which determine which of the syntactically permitted collections of finer relationships are valid.

References

1. Barr, M., Wells, C. Category Theory for Computer Scientists. Englewood Cliffs, NJ: Prentice-Hall 1990
2. Baclawski, K., Dimovici, D., White, W.: A categorical approach to database semantics. Math Struct Comput Sci 4, 147–183 (1994)
3. Batini, C., Ceri, S., Navathe, S.B.: Conceptual Database Design: an Entity-Relationship Approach. Menlo Park, CA: Benjamin/Cummings 1992
4. Dampney, C.N.G., Johnson, M.S.J.: Application of Consistent Dependency to Corporate and Project Information Models. In: Papazoglou, M. (ed.) OOER′95 Object-Oriented and Entity-Relationship Modelling, 19th International Conference Lecture Notes in Computer Science 1021, Berlin Heidelberg New York: Springer 1995
5. Dampney, C.N.G., Johnson, M.S.J., Dazeley, P., Reigh, V.: A Higher-Order 'Commuting Loop' Structure that Supports Very large Information System Data and Process Architecture. IFIP TC8 Working Conference, Bond University, Gold Coast QLD Australia, May, 1994
6. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, Second Edition. Menlo Park, CA: Benjamin Cummings 1994
7. Gane, C., Sarson, T.: Structured Systems Analysis: Tools and Techniques Improved Systems Technologies, Inc, 1977
8. Islam, A., Phoa, W.: Categorical models of relational databases I: fibrational formulation, schema integration. In: Hagiya, M., Mitchell, J.C. (eds.) Theoretical Aspects of Computer Software International Symposium TACS′94. LNCS 789, pp. 618–641, Berlin Heidelberg New York: Springer 1994
9. Johnson, M.S.J, Dampney, C.N.G.: On the Value of Commutative Diagrams in Information Modelling. In: M. Nivat, C. Rattray, T. Rus, G. Scollo (eds) Algebraic Methodology and Software Technology (AMAST′93) Workshops in Computing, London: Springer 1993
10. Lippe, E., ter Hofstede, A.: A Category Theoretical Approach to Conceptual Data Modelling. RAIRO Theor Inform Appl 30(1), 31–79 (1996)
11. Rosebrugh, R., Wood, R.J.: Relational Databases and Indexed Categories. In: Proceedings of the 1991 Category Theory Meeting, American Mathematical Society, 1991
12. Simsion, G.: Data Modelling Essentials: Analysis, Design and Innovation. New York: Van Nostrand Reinhold 1993
13. Ullman, J.D.: Principles of Database and Knowledge-Base Systems (Vol. 2) Computer Science Press, 1989