Ontology engineering, universal algebra, and category theory *

Michael Johnson[1] and Robert Rosebrugh[2]

[1] Department of Computer Science
Macquarie University
mike@ics.mq.edu.au
[2] Department of Mathematics and Computer Science
Mount Allison University
rrosebrugh@mta.ca

Abstract. In this chapter we review a category theoretic approach to ontology engineering. Using ideas from universal algebra, we carefully distinguish presentations of ontologies from the ontologies themselves. This leads to a precise notion of "view", and views can be used both to create new ontologies incorporating existing ontologies while recognising their common classes and relations, and to develop interoperating ontologies. Interoperating ontologies are separate but linked ontologies with the property that systems developed under each ontology can interoperate without changing the system and with only a small amount of front-end interoperations code.
Keywords: View update, semantic data model, category theory, ontology, interoperation.

1 Introduction

Ontology engineering is most often used to refer to the (indeed vitally important) process of constructing ontologies.

Thus it was once with traditional engineering — engineers were concerned with building essentially unique artifacts: bridges, roads and buildings for example. The engineer had learnt from other examples and would bring general concepts, known solutions, rules-of-thumb, and scientific calculation to the particular problems presented by a new site. Sometimes the engineer could effectively reuse a design, adjusting only a few parameters (height of the towers, length of a span etc) because a new site was sufficiently like an old site, but essentially each new artifact called for a newly engineered solution. In contrast much modern engineering is fundamentally about developing interoperations among extant systems — telecommunications is almost by definition thus, and most modern manufacturing depends very significantly on planning and managing the interactions between known systems.

* Research partially supported by grants from the Australian Research Council and NSERC Canada.

The irony of ontology engineering needing to focus on constructing individual ontologies (whether small and domain specific (see examples in [2]) or extensive and intended to establish standards over a wider field [3]) is of course that ontologies themselves were introduced to aid system interoperability. A good theory and detailed processes for *ontology interoperability* will significantly aid the development of new ontologies incorporating old ones. Then extant ontologies can be used to support systems interoperation even when the distinct systems are based themselves in separately developed ontologies. This was the goal of the Scalable Knowledge Composition group's *algebra of ontologies* [11].

This chapter follows and further develops [9] in drawing ideas from categorical universal algebra, a field which might be viewed as ontology engineering for mathematics, and using them to support ontology engineering in a manner that leads naturally, and mathematically, to support for interoperations among ontologies.

The ideas presented here have a strong theoretical foundation in the branch of mathematics called *category theory* (see the chapter by Healy), and they have been developed and tested in a range of industrial applications over the last fifteen years.

2 Representing ontologies

Formal ontologies are at least "controlled vocabularies" and can take many forms ranging from glossaries to first order logical theories [12]. Often ontologies are expressed as trees or directed graphs because *subsumption* is such an important (real-world) relationship and we are used to representing subsumption relationships as directed edges. In fact, subsumption is a particular example of a function (to each instance of the subclass, there corresponds a particular instance in the superclass), and graphically or not, ontologies depend fundamentally on being able to represent functions.

Of course, it is largely unimportant how we represent ontologies because there is substantial software support for ontology engineering. It is vital that a proposed ontology is given via a precise representation, but then that representation can be compiled into other forms.

In this chapter we will consider ontologies represented as categories. This will have several advantages for us, including the use of the technical tools of category theory (see the chapter by Healy) to support our analysis of ontologies, the use of categorical logic (see the chapter by Vickers) to make clear the link between ontologies as categories and ontologies as logical theories, and the graphical representation and reasoning of category theory to easily allow fragments of ontologies to be represented in their more usual form as graphs or trees.

As an example we present a fragment of an ontology for air traffic control systems. For those not used to category theory it will suffice to contemplate the diagram (Figure 1) and we will explain the categorical constructions in that diagram in logical terms. For ease of exposition we have focused upon the part of the ontology related to aircraft planning a landing.

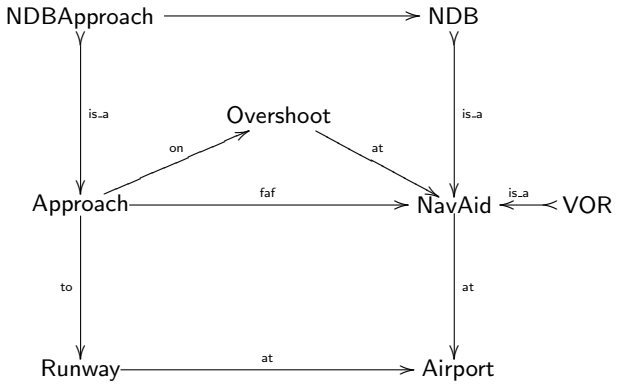

Fig. 1. A fragment of an air traffic control ontology

Nodes in the directed graph shown in Figure 1 are the main subject matter of the ontology. In this case we are concerned with airports, runways, navigational aids (VOR, which stands for VHF Omni-Range, and NDB, which stands for Non-Directional Beacon), and aircraft approaches to airports. As noted by Gruber [6], ontologies are typically represented using classes, relationships (between classes) and attributes (of instances of classes). In Figure 1 classes are represented by nodes and functional relationships by arrows. Attributes, while important and easily represented in our category theoretic approach, are suppressed to keep the diagram simple.

At this point it is worth making three remarks about our use of functional relationships.

First, all traditional (many-to-many) relationships can be represented as a "span" of functional relationships. For example Approach might be a relationship between Runway and NavAid (some navigational aids can be used as final approach fixes for particular runways). Here that is expressed by the directed edges to and faf (final approach fix). As is often the case, an attempt to "tabulate" the (many-to-many) relationship between Runway and NavAid in fact records a class that has semantic significance and which might itself be explicitly recorded as a class in other related ontologies. So, choosing to represent only functional relationships doesn't limit our representational power, and may (in practice, does often) assist in identifying meaningful classes.

Second, because ontologies are intended to be more abstract than data models, what in many ontologies appears as a functional relationship in fact may be only a partial function. If one were to say for example that an airport has a radio frequency used for approaches, one would be largely correct. And it is certainly worth recording that an airport may have such a frequency, and that that relationship is in general functional, but strictly the function would be partial since there are uncontrolled airports. Such partiality most often arises with optional

attributes and can be dealt with by introducing a (tabulated as just discussed) relation. How one does this turns out to include some theoretical surprises, and we will discuss it in more detail in Section 8.

Third, is_a relationships, among others, are denoted here by edges indicated $\twoheadrightarrow$. Those edges are intended to be realised not just as functions, but as injective (one-to-one but not necessarily onto) functions. This makes sense since if every instance of X is an instance of Y, then we certainly expect that given an instance of X we have a corresponding instance of Y (ie we have a function), but furthermore no two distinct instances of X would correspond under the is_a relation to the same instance of Y (ie the function is injective). It is also valuable to be able to specify that other (functional) relationships are also required to be injective. For example, the function faf specifies, for an approach to a particular runway, which navigational aid provides the final approach fix. We will return to the specification of $\twoheadrightarrow$ arrows in Section 3.

Remembering that Figure 1 is a representation of part of a category, there are other constraints. For example, in a category, arrows can be composed and so diagrams do, or do not, commute (see again the chapter by Healy). In our example, both rectangles commute. To check the commutativity of the lower rectangle we note simply that an Approach uses as final approach fix a NavAid which is located at an Airport, and that the approach is to a Runway which must be at the same airport as the navigational aid. On the other hand the triangle does not in general commute: An Approach has associated with it an Overshoot which indicates which NavAid an aircraft should go to in the case of failing to land after the particular approach. The overshoot navigational aid will not usually be the aid that was used as the final approach fix for the approach.

It is important to emphasise that whether or not a diagram commutes is a question of real-world semantics — a question that should be asked, and a question whose answer should be recorded as part of the ontology. Further examples of the importance of specifying commutative and not necessarily commutative diagrams are given in [8].

Finally there are other, also categorically and logically expressible, interactions between the nodes shown in Figure 1. The is_a arrows into NavAid tell us that all instances of VOR and NDB are NavAids. There is nothing in the diagram that tells us that the "subobjects" are disjoint from each other, nor is there anything to say whether or not there can be NavAids which are of other types. If, as is the case in the real ontology, NavAid is known to be the *coproduct* of the two subobjects, then that ensures that both these conditions are satisfied. In set theoretic terms the coproduct requirement ensures that NavAid can be obtained as the disjoint union of VOR and NDB. Similarly NDBApproach can be computed as a *pullback* in set theoretic terms, which in this case is the inverse image of the inclusion of NDB's in NavAid's along faf. It selects those Approaches whose NavAid is an NDB.

Note that although NavAid and its two is_a arrows can be computed from other classes, it needs to be presented in Figure 1 as it is the codomain of two further arrows. On the other hand the ontology would not be changed at all if

NDBApproach were left out. It, and its two arrows, and the commutativity of its rectangle, are all determined and can be recomputed as needed. This flexibility is at the heart of ontology interoperation discussed in Section 5.

3 Presenting ontologies

In the last section we noted (by example) that an ontology may seem to take a different form merely because classes may not be included although they may still be fully determined by other classes and relationships which are present. This is the essence of the semantic mismatch problem which has made system interoperability so difficult and has motivated many of the developments of ontologies. Probably the potential complications are obvious to those who have worked with independently developed but partially overlapping ontologies: each of the ontologies contains enough information to model the application domain, but they seem hardly comparable because classes, often sharing the same name, can be two different specialisations of a more general class that doesn't appear in either ontology, and because each ontology seems to contain important classes that are absent from the other.

In this section we briefly review mathematical presentations and the forms of category theoretic presentation that the authors have found empirically sufficient for ontology engineering.

A *presentation* is a specification of the generators and axioms that are required to determine a mathematical object. Common examples include in group theory presentations of a group, or in the theory of formal languages presentations of a language. A presentation is important because it gives a precise and usually finite specification which suffices to fully determine the object. Yet presentations are not usually the subject of mathematical study since a given mathematical object will usually have many different presentations.

Categories are presented using *sketches* [1]. At its most basic, a sketch consists of a directed graph (like Figure 1) together with a set of diagrams in the graph (pairs of paths of arrows with common start and endpoint) which are intended to be the commutative diagrams. The category presented by the sketch is simply the smallest category, generated by the graph, subject to the commutativity of the diagrams (and hence of all others that follow logically from those diagrams).

More generally a *mixed sketch* is a sketch together with sets of cones and cocones (see the chapter by Healy) in the graph which are intended to be limit and colimit cones. The category presented by the (mixed) sketch is simply the smallest category with finite limits and finite colimits, generated by the graph, subject to the commutativity of the diagrams and to the requirement that the cones and cocones do indeed form limit cones and cocones.

The formal development of the theory of (mixed) sketches is not important for us here but one empirical observation is important: over a wide range of practical studies finite limits and finite coproducts have sufficed to specify ontologies. So, we will limit our attention to these kinds of mixed sketches. Furthermore, it turns out that finite limits and finite coproducts are sufficient to support, via

categorical logic (see the chapter by Vickers) a large fragment of first order logic. This fragment is sufficient to model all usual "queries" for example. It is a powerful tool for specifications, and easily supports, for example, the specification of monic arrows which are modelled by injective functions. Indeed, an arrow is monic exactly if its pullback along itself can be obtained with equal projections.

Having settled on the presentations we use, we now note in the strongest of terms: An ontology is not its presentation. If you accept our claim that finite limits and finite coproducts should be used in the category theoretic definition of an ontology, then whenever you ask to see an ontology you will be shown a presentation. This is simply because the category with finite limits and finite coproducts specified by the presentation will in general be infinite, so can't reasonably be shown to you. But, as with other mathematical presentations, the object of study is not the presentation, but rather the (infinite) category.

When one takes this point of view many of the difficulties of independently developed but overlapping ontologies disappear. If two such ontologies really do capture the same real-world domain then the independent *presentations* have the incompatibilities discussed at the beginning of this section, but the ontologies that they present will be the same.

Of course, other ontologists have recognised this point in their own frameworks. In particular, taking an ontology as a first order logical *theory* (rather than a collection of logical sentences) corresponds to taking the ontology as a category rather than as a presentation. Nevertheless, it is apt to be forgotten when working with tools that support and indeed display only finite fragments.

4 Views versus sub-ontologies

An immediate benefit of recognising ontologies as categories rather than presentations comes with the definition of views. In many practical treatments a view of an ontology is in fact a sub-presentation. When such a view exists, it will behave well. Certainly a subpresentation is a view. But there are many views of ontologies whose basic classes include some which do not occur in the presentation.

Instead, a view of on ontology $\mathcal{O}$ should be a presentation $\mathbb{V}$ together with a mapping of that presentation into the entire ontology $\mathcal{O}$, not just into the presentation for the ontology $\mathcal{O}$.

The idea here is worth emphasising: A view of an ontology should be given just like any other ontology would be, via a presentation $\mathbb{V}$, but when it comes to instances the ontology generated by that presentation should be populated by instances determined by the mapping of the presentation into the ontology $\mathcal{O}$.

Incidentally, in the category theoretic treatment, a mapping of a presentation $\mathbb{V}$ into an ontology $\mathcal{O}$ is the same as a functor, preserving limits and coproducts, between the ontology $\mathcal{V}$ generated by the presentation $\mathbb{V}$ and the ontology $\mathcal{O}$. If in a particular application the ontology $\mathcal{O}$ has sets of instances associated with its classes, then the view is obtained for any class V by following the functor to

$\mathcal{O}$, obtaining a class in $\mathcal{O}$, and then seeing what set of instances is associated with that class.

We choose to work always with views and to eschew the use of subontology except for the very simple cases which arise as a result of subpresentations.

5 Interoperations

The last section dealt carefully with views and subpresentations because some particularly forward looking work in the 1990s was intended to develop interoperations for ontologies via an *algebra of ontologies* [11]. The algebraic operations were based on the "intersection" of ontologies, and we claim that the notion of *view* introduced in the preceding section supports an appropriate generalisation of the notion of intersection.

The idea is as follows.

First, let's review the familiar notion of intersection. Given two structures A and B their *intersection* is the largest structure C which appears as a substructure of both A and B. Diagrammatically $A \leftarrow C \rightarrow B$, where the arrows are inclusions and C is the largest structure that can be put in such a diagram. Thus if we seek the intersection of two presentations of two ontologies the construction is fairly clear: We merely take the set-theoretic intersection of the classes of the ontologies, and the set theoretic intersection of the relations between those classes.

Of course, the foregoing is too naive. Two different ontologies may have two common classes X and Y, and in each ontology there may be a functional relation $X \rightarrow Y$, but the two relations may be semantically different. Perhaps we could require that the relations carry the same name when we seek their intersection, but this brings us back to the very problem ontologies are intended to control — people use names inconsistently.

Instead, we take seriously the diagram $A \leftarrow C \rightarrow B$, developed with insightful intervention to encode the common parts of A and B and record that encoding via the two maps. Thus an $f : X \rightarrow Y$ will only appear in C when there are corresponding semantically equivalent functions in A and B, whatever they might be called.

The requirement to check for semantic equivalence is unfortunate but unavoidable. Mathematically structures can be linked by mappings provided the mathematical form that they take does not contradict the existence of such a mapping, but whether such mappings are meaningful is a semantic question that requires domain knowledge.

Now we need to note that so far we have only dealt with presentations. If we want an appropriate generalisation of intersection for ontologies we need to recognise that two ontologies can have common classes. The commonality only becomes apparent when one moves from the presentations, to the ontologies. For example, two ontologies might both have a class called Product, but if the ontologies were developed for different domains those classes are very unlikely to be semantically equivalent. Nevertheless, it might happen that the ontologies

do both deal with products of a certain type. To find an "intersection" it will be necessary to specialise both of the Product classes, perhaps by restricting them to products with certain attributes (likely different in the two different ontologies) or that have certain relationships with other classes (again likely different in the two different ontologies). In our experimental work such situations arise frequently, but with the limits and coproducts that are available in the two ontologies (represented as categories with finite limits and finite coproducts) we can analyse the specialisations and determine the corresponding classes in the two ontologies. Thus, if the two ontologies are called $\mathcal{O}$ and $\mathcal{O}'$ we develop a view $\mathbb{V}$ together with mappings into $\mathcal{O}$ and $\mathcal{O}'$ obtaining $\mathcal{O} \leftarrow \mathbb{V} \rightarrow \mathcal{O}'$.

The largest such $\mathbb{V}$ might be viewed as the "generalised intersection" of $\mathcal{O}$ and $\mathcal{O}'$.

In fact, in our empirical work we can rarely be confident that we have obtained the largest such $\mathbb{V}$. No matter, $\mathbb{V}$ together with its mappings provides an explicit specification of identified common classes in the two ontologies that should be kept synchronised if we expect the ontologies to interoperate. In practical circumstances two industries will have specific intentions about how they will interoperate, and so there is guidance as to which general areas of their ontologies will need to be synchronised. There may be other commonalities (for example, Person might be semantically equivalent in the two ontologies), but if as industries they keep those parts of their operations distinct we need not necessarily support interoperations on those commonalities.

It is worth emphasising here that views are better than subpresentations, and ontologies as categories with finite limits and finite coproducts are better than mere presentations of ontologies, because, at least in our experience, only in very fortuitous circumstances will ontologies be able to be "linked" via classes which happen to appear already in their presentations.

6 Solving view updates

Although this chapter has dealt mostly with ontologies, rather than with the applications of ontologies (often databases of some kind) that store the instances of the classes that occur in the ontology, it is important to consider carefully the interaction between views and instances.

In one direction the interaction is straightforward. As noted in the previous section, a view's classes should be populated with the instances from the corresponding classes in the ontology. In category theoretic terms, the assignment of instances to classes is a functor from the ontology (viewed as a category) to a category of (usually finite) sets. Then a view, being not simply a presentation $\mathbb{V}$ but also a functor between the ontology $\mathcal{V}$ determined by that presentation, and the fundamental ontology $\mathcal{O}$, can be composed with the instances functor $\mathcal{O} \rightarrow \mathbf{set}$ to yield an instances functor $\mathcal{V} \rightarrow \mathbf{set}$.

Thus modifying the recorded instances of an ontology automatically modifies the instances of any view of that ontology.

The reverse direction presents significantly more complications. In particular, if one were to treat a view and its instances as if it were simply an ontology, and one were to modify (update) the recorded instances of the view, it's a very significant problem to determine how best to correspondingly modify the recorded instances of the ontology. In database theory this has been known as "the view update problem" and has been the subject of research for nearly thirty years.

The difficulty of the view update problem is easily underestimated when one concentrates on the classes, as we so often do when working with ontologies, and neglects the relations between classes. But fundamentally the information about an instance of a class is contained in the way that that instance is related to other instances of other classes, or indeed to attributes. Thus, when one introduces or deletes an instance from a class in the view, an instance must be likewise introduced or deleted from the corresponding class in the ontology. But what happens to the relationships that that instance might participate in? In some cases the relationships will also be present in the view, and then we can see what we should do to them in the ontology, but inevitably there will be relationships which are not present in the view but which need to be modified in the ontology. How should we modify them if we want to update the instances of the ontology to match the instances of the view?

Over the years there have been many solutions proposed for the view update problem. We won't review them here, but we will point out that there is one solution suggested naturally by the category theoretic approach — category theory makes extensive use of so-called *universal properties* (properties like those used to define limits and colimits in the chapter by Healy), and there is a natural such property, well-known in category theory, which can provide arguably optimal solutions to view update problems. The required property is the existence of cartesian and op-cartesian arrows, and the details are presented in the authors' [10].

Rather than embarking on the category theoretic details, we will proceed now to show how solutions to view update problems, when they exist, can be used to develop ontology interoperation, and hence ultimately to aid ontology engineering in the sense discussed in the introduction — engineering new ontologies by carefully developing interoperations among existing ontologies.

7 Interoperations with instances

For many purposes, including an algebra of ontologies, and for many ontologists, the identification of a common view in the manner of Section 5 suffices. A new ontology incorporating both extant ontologies and respecting their common views can be calculated by taking the colimit of $\mathcal{O} \leftarrowtail \mathcal{V} \twoheadrightarrow \mathcal{O}'$ in the category of categories with finite limits and finite colimits. This corresponds to Healy's "blending of theories". Nevertheless, we can ask for more.

Many ontology projects, including both BizDex [3] and aspects of e2esu [5], seek to develop interoperations based on independently developed domain-based ontologies. Interoperations in these cases mean interoperations at the instance

level, and as we saw in the preceding section, interoperations at the instance level are more subtle than the colimit ontology would suggest.

We say that ontologies $\mathcal{O}$ and $\mathcal{O}'$ interoperate via a view $\mathbb{V}$ when in the diagram $\mathcal{O} \leftarrow \mathbb{V} \rightarrow \mathcal{O}'$ the view update problems have both been solved. In such a case the ontologies support interoperating systems. Suppose that we have two information systems I and I' based respectively in the ontologies $\mathcal{O}$ and $\mathcal{O}'$, then the systems can interoperate as follows. Suppose that a change is made to the information stored in system I. Immediately that results in a change to the $\mathbb{V}$ view of I. Because we have a solution to the view update problem for the $\mathbb{V}$ view of I' that change in the view can be "propagated" to the information system I'. Similarly, changes in the information stored in I' are propagated via the common view to I. The result is that the two systems remain synchronised on their common views while they operate independently and, apart from the view update code, without any modification to the original stand-alone information systems.

A detailed example for e-commerce systems is presented in [7]. That paper also points out that this "full-duplex" interoperability is in fact stronger than is often required in business. Instead, a "half-duplex" approach to interoperability often suffices. This takes advantage of the empirical observation that for particular interoperating businesses information only flows one way on certain classes, and the other way on other classes, and these limited flows reduce the difficulty of solving the view update problems.

We reserve the term *interoperating ontologies* for cases like those discussed in this section where view update problems have been solved, at least in the half-duplex sense, and so the $\mathbb{V}$-linking of ontologies yields a corresponding and effective linking of extant systems constructed using those ontologies.

8 Nulls and partial functions

It is notoriously difficult to precisely distinguish ontologies from data models. Even Gruber's definition [6], in common with other treatments, can only broadly distinguish ontologies by discussing their expected independence of data-structures and implementation strategies. He also notes that primary keys are the kind of thing that one would *not* expect to find in an ontology.

One only half-facetious proposal put forward by the authors is that ontologies are, at least often in practice, those data models in which all functional relationships are partial. This does correspond with remarks about primary keys, since a defining property of primary keys is that they are mandatory attributes. But more than this, ontologies frequently provide a controlled vocabulary that indicates what *might* be said about instances. Thus attributes are optional, or in other words, attributes are always allowed to take null-values.

This suggests that in a theory of ontologies it is very important to determine how one represents null-values or partial functions.

There are two widely used representations, both in databases and in category theory. In one, a partial function is modelled by including an explicit null-value.

Thus a partial function $X \nrightarrow A$ is represented by total function $X \rightarrow A + 1$ with the same domain X, but whose codomain is augmented by summing it with a single `null` value. When the function is undefined on a particular $x \in X$ the value at x for the total function is by definition the extra or `null` value of the codomain $A + 1$. In the other approach a partial function $X \nrightarrow A$ is modelled by a relation $X \leftarrowtail X' \rightarrowtail A$ in which X' should be viewed as the subobject of X for which the function is defined. Surprisingly a careful analysis of these two approaches shows that in the context of instances, viewed as categories of models, they are not equivalent. See the discussion of ontological problems with "not" discussed in the chapter by Vickers.

Although the explicit use of null-values is probably the most widespread representation of partial functions in databases, the authors choose to represent partial functions using the relation approach. Using this approach we accept ontologies in which explicit functional relations are drawn as arrows, but interpret those arrows $X \rightarrow A$ as an abbreviation standing for $X \leftarrowtail X' \rightarrowtail A$. This is in contrast to our treatment of data models in which an arrow $X \rightarrow A$ is interpreted as a total function. When it comes to making precise category theoretic calculations in ontologies the spans $X \leftarrowtail X' \rightarrowtail A$ need to be explicitly included.

9 Universal nulls

One of the great advantages of interpreting functions in ontologies as partial functions is that view update problems much more often have straightforward solutions. Perhaps this is easy to see. If a view includes a class, but not an attribute of that class which is present in the ontology, then solving a view update problem will be very difficult. After all, new instances of the class in the view don't come with an attribute value, but in the ontology, at least if functions are interpreted as total functions, each new instance needs to be associated with a particular value for the attribute. Recall that view update problems are solved using category theoretic universal properties. When a function needs to take a value, but no value is in any sense canonical, then there is very little chance of finding a universal solution.

One might reasonably expect that when functions are allowed to be partial, the null-value will in some sense be canonical. Certainly, assigning the null-value in cases where no other value is determined by the view is the minimal change. Unfortunately, an explicit null-value is not category theoretically distinguishable from any other value of an attribute, and so the difficulty in finding a universal solution remains.

Happily, using the relation approach from the previous section, assigning a null-value to an instance x in the relation $X \leftarrowtail X' \rightarrowtail A$ simply means not including x in the subobject of defined values X'. This is again the minimal change, but this time it is also minimal with respect to natural transformations among the set-valued functors which keep track of the assignment of instances

to classes. It turns out that this minimality is exactly what is required to provide a universal solution for the view update problem.

10 Conclusion

Developing interoperating systems is one of the most important uses of ontologies. Over recent years category theory has been used to develop new approaches to ontology presentation, and new solutions to view update problems. View update solutions can be used to engineer systems interoperation.

In this chapter we have studied the relationship between category theoretic specification and ontologies. We have noted how views can be used in a general way to support calculating the colimit of two ontologies so as to create a new ontology which includes them both and recognises their commonalities. We have shown further how solving view update problems can lead to systems interoperation without a need to modify the basic systems. In this latter case we call the ontologies *interoperating ontologies*. We have noted briefly how view update solutions can be aided by the limitations of half-duplex interoperation, and by the very common if often inexplicit use of partial functions in ontologies.

Over all we have found that new and mathematically precise treatments of difficult problems in the foundation of ontologies and information systems support ontology interoperation.

It's always pleasing when theoretical developments yield practical advantages and new insights as well as stronger theory.

References

1. M. Barr and C. Wells. *Category theory for computing science*. Prentice-Hall, second edition, 1995.
2. Elena Paslaru Bontas, Christoph Tempich. Ontology Engineering: A Reality Check. In R. Meersman and Z. Tari et al, The 5th International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE2006), (LNCS 4275), 836–854. Springer, 2006.
3. BizDex is an e-Business framework that incorporates common Standards. See `http://www.standards.org.au/cat.asp?catid=37&contentid=73` (accessed January 2, 2008)
4. John Davies, Rudi Studer and Paul Warren (eds). Semantic Web Technologies: Trends and Research in Ontology-based Systems, 326pp, 2006.
5. End-to-End Service Utility (E2ESU). See `http://www.e2esu.eu/public/` (accessed November 13, 2007)
6. Tom Gruber. Ontology. To appear in the Encyclopedia of Database Systems, Ling Liu and M. Tamer Özsu (eds), Springer-Verlag, 2008.
7. Michael Johnson. Enterprise Software with Half-Duplex Interoperations. In press for I-ESA, Bordeaux, 2006, Lecture Notes in Computer Science.
8. Michael Johnson and C.N.G. Dampney. On the value of commutative diagrams in information modelling. Springer Workshops in Computing, eds Nivat et al, 1994, 47–60, Springer, London.

9. Michael Johnson and C.N.G. Dampney. On category theory as a (meta) ontology for information systems research. In Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS2001) 59–69, ACM Press, 2001.
10. Michael Johnson and Robert Rosebrugh. Fibrations and Universal View Updatability. *Theoretical Computer Science* 388, 109–129, 2007.
11. Prasenjit Mitra and Gio Wiederhold. An Ontology-Composition Algebra. In S.Staab, R.Studer (eds). Handbook on Ontologies, Springer International Handbooks on Information Systems, 93–113 2004.
12. Barry Smith and Christopher Welty. FOIS Introduction: Ontology—towards a new synthesis. In Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS2001) 3–9, ACM Press, 2001.