Semantics for Interoperability
relating ontologies and schemata

Trevor Bench-Capon, Grant Malcolm, and Michael Shave

Department of Computer Science, University of Liverpool
PO Box 147, Liverpool L69 3BX, UK

Abstract. We present a formal approach to ontologies that supports reuse and sharing of information through relationships between compatible conceptualisations. This general approach is extended to database schemata, which allows us to characterise precisely the relationships between schemata and ontologies, and to give some foundations for interoperability of heterogeneous information systems. Our main results give a correctness criterion for relationships between schemata, and show that compatible schemata can be combined to provide an appropriate language for interoperation at an abstract, conceptual level.

1 Introduction

Through the WWW information has become accessible in ever increasing quantities. In consequence the potential benefits of information reuse have become increasingly important, as is recognised by initiatives such as the Semantic Web [1]. The major obstacle to reuse is that information is produced according to the varying conceptualisations of the many designers and the many purposes for which it is gathered and represented. It thus embodies a number of assumptions reflecting the background, experience and needs of its producers. For reuse to be possible these assumptions must be made as available as the information itself. In the case of fully specified information systems, most typically relational databases, the existence of schemata provide the required basis for reuse, and interoperability of databases through integration schemata is now commonplace. For less tightly specified information sources, these assumptions very often remain implicit. The response to this has been the development of ontologies, intended to supply formal expressions of the conceptualisations of a domain of knowledge. Even when an ontology is available, however, the need to recast information developed according to one ontology in a form usable by a system developed using a different ontology remains. The solution is the use of a shared ontology which captures what is common between the two conceptualisations, but this may still need procedural information to render it usable by an application: this, to provide parallels with database systems, we refer to as an integration schema. In this paper we attempt to make precise the relationship between ontologies and schemata, so as to provide a formal basis for the reuse of information, regardless of the type of system from which it is derived.

In Sect. 2 we give formal description of ontologies, and in Sect. 3 we provide a compatible description of schemata, and show how this can be used to describe relations between schemata and between schemata and ontologies. For reasons of space, we give only brief motivations for definitions, and omit proofs; these proofs, more extensive motivations, and some technical details we have omitted here for brevity's sake, are given in a technical report [2].

2 Specifying and Relating Ontologies

In this section we give a formal account of specifications of ontologies, and summarise some results from [3] that provide support for knowledge reuse.

Throughout this paper, we assume a given collection, D, of data types with names (always in italics) such as *String*, providing standard data representations for attribute values. Note that [3, 2] take a more general approach to the use of data types in specifying both ontologies and schemata.

Definition 1. *An* **ontology signature***, or just* **signature** *for short, is a pair* (C, A)*, where* $\mathsf{C} = (C, \leq)$ *is a partial order, called a* **class hierarchy***, and* A *is a family of sets* $A_{c,e}$ *of* **attribute symbols** *for* $c \in C$ *and* $e \in C + D$*. This family of sets is such that* $A_{c',e} \subseteq A_{c,e'}$ *whenever* $c \leq c'$ *in* C *and* $e \leq e'$ *in* $C + D$*.*

(We write $C+D$ for the disjoint union of classes and data types; i.e., an attribute might take values in either a class or a data type.)

Example 1. The following is an ontology that might be used by a second-hand car dealer. There are four classes, Car, Estate-Car, Saloon-Car and Model. Each class is followed by an indented list of its attributes and their result types.

```
Car
    colour : {white, red, blue}
    model : Model
    price : Pounds

Estate-Car < Car
    rear-space : SquareMetres

Saloon-Car < Car
    hatchback : {y, n}

Model
    name : String
    manufacturer : {maker1, maker2}
    photo : Gif
```

The class Car has three attributes, e.g., model, which takes values in the class Model (i.e., model $\in A_{\mathsf{Car},\mathsf{Model}}$). The class Estate-Car has four attributes: three inherited from Car plus rear-space, which takes values in a data type *SquareMetres*.

The semantics of signatures is given by *models*: essentially, a model M interprets each class name c as a set M_c, and each attribute $\alpha \in A_{c,e}$ as a function M_α of the appropriate type ($M_c \to M_e$, if e is a class name; $M_c \to D_e$, where D_e is the set of appropriate data values, if e is a data type), subject to certain monotonicity requirements (see [3, 2] for details). While models provide a notion of interpretation, the following gives a notion of translation:

Definition 2. *An* **ontology signature morphism** $\chi : (\mathsf{C}, A) \to (\mathsf{C}', A')$ *is a pair* $\chi = (f, g)$*, where* $f : \mathsf{C} \to \mathsf{C}'$ *is a morphism of partial orders, and* g *is a family of functions* $g_{c,e} : A_{c,e} \to {A'}_{f(c),f(e)}$*, where* $f(e) = e$ *if* e *is a data type in* D*, such that if* $c \leq c'$ *and* $e \leq e'$ *then* $g_{c',e}(\alpha) = g_{c,e'}(\alpha)$ *for all* $\alpha \in A_{c',e}$*.*

Interpretations can work backwards along translations, in the following sense: given a morphism $\chi : (\mathsf{C}, A) \to (\mathsf{C}', A')$, and given a (C', A')-model M, we define the (C, A)-model χM by

– $(\chi M)_c = M_{f(c)}$ for $c \in C$, and
– $(\chi M)_\alpha = M_{g_{c,e}(\alpha)}$ for $\alpha \in A_{c,e}$.

An ontology consists of a signature together with some axioms that constrain the values of attributes (see Example 2 below). A **morphism** of ontologies $\chi : \mathcal{O} \to \mathcal{O}'$ is a signature morphism such that whenever M is a model of $\mathcal{O}'$ (i.e., M satisfies the axioms of $\mathcal{O}$) then χM satisfies the axioms of $\mathcal{O}$. Intuitively, this means that $\mathcal{O}'$ has more constraints than $\mathcal{O}$ has.

Example 2. As an example, we give an ontology for models of cars that might be used for a buyers' guide:

Model
  name : *String*
  type : $\{estate, saloon\}$
  hatchback : $\{y, n, n/a\}$
  picture : *Gif*
  rear-space : *SquareMetres*
  manufacturer : Motor-maker
  price : *Pounds*

Car
  colour : $\{white, red, blue\}$
  model : Model

Motor-Maker
  name : $\{maker1, maker2\}$
  dealer : Dealer

Dealer
  name : *String*
  address : *String*

Axioms
hatchback(M) $= n/a$ if type(M) $= estate$

The axiom constrains the value of hatchback depending on the type of model.

Morphisms express relationships between ontologies; the 'cocompleteness theorem' in [3] states that ontologies can be combined without duplicating shared components:

Theorem 1. *The category of ontologies and their morphisms is cocomplete.*

In the terminology of category theory, a combination of related ontologies produces a 'colimit'; the following illustrates what this means:

Example 3. Consider the following ontology, which provides a common subcomponent for the ontologies of Examples 1 and 2.

Car
  colour : $\{white, red, blue\}$
  model : Model

Model
  name : *String*
  manufacturer : $\{maker1, maker2\}$
  photo : *Gif*

There is an inclusion from this ontology (call it $\mathcal{O}$) to the ontology (call it $\mathcal{O}_1$) of Example 1. This gives us $\chi_1 : \mathcal{O} \to \mathcal{O}_1$. To relate $\mathcal{O}_1$ to the ontology (call it $\mathcal{O}_2$) of Example 2, we construct a morphism $\chi_2 : \mathcal{O} \to \mathcal{O}_2$ as follows. The class Car and its attributes are simply included in $\mathcal{O}_2$. The class Model and its attribute name are likewise included, and the attribute photo is mapped to the attribute picture in $\mathcal{O}_2$. The attribute manufacturer is treated in a more complex manner: it is mapped to the *compound* attribute manufacturer;name in $\mathcal{O}_2$, i.e., take the manufacturer attribute, giving a value in the class Motor-Maker, then take the attribute name of that class, giving a value in $\{maker1, maker2\}$.

In the colimit of this relation the attribute photo represents both the photo attribute of $\mathcal{O}_1$ and the picture attribute of $\mathcal{O}_2$. This is because these two attributes are the image of photo in $\mathcal{O}$ under χ_1 and χ_2 respectively, so this attribute is shared between the two ontologies, just as all the attributes of the class Car are shared. However, there are two manufacturer attributes in the colimit; one representing the attribute of the same name from $\mathcal{O}_1$, and the other (which we call manufacturer2) representing the attribute from $\mathcal{O}_2$. The morphisms χ_i state that manufacturer and manufacturer2;name should be the same, so in the colimit there is an axiom that constrains all models to treat them in the same way:

```
manufacturer(M) = name(manufacturer2(M)) .
```

The colimit of our two related ontologies is given in full below.

Car
- colour : $\{white, red, blue\}$
- model : Model
- price : *Pounds*

Estate-Car < Car
- rear-space : *SquareMetres*

Saloon-Car < Car
- hatchback : $\{y, n\}$

Dealer
- name : *String*
- address : *String*

Model
- name : *String*
- type : $\{estate, saloon\}$
- hatchback : $\{y, n, n/a\}$
- picture : *Gif*
- rear-space :
- manufacturer : $\{maker1, maker2\}$
- manufacturer2 : Motor-maker
- price : *Pounds*

Motor-Maker
- name : $\{maker1, maker2\}$
- dealer : Dealer

Axioms

manufacturer(M) = name(manufacturer2(M))

hatchback(M) = n/a if type(M) = *estate*

In general, colimits combine the constraints of the related ontologies; the following gives a sufficient condition for these constraints to be consistent (i.e., satisfiable):

Proposition 1. *Let $\chi_i : \mathcal{O} \to \mathcal{O}_i$ for $i = 1, 2$ and let M_i be a model of $\mathcal{O}_i$ for $i = 1, 2$ such that $\chi_1 M_1 = \chi_2 M_2$; then the colimit of χ_i has a model, and so $\mathcal{O}_1$ and $\mathcal{O}_2$ are compatible over $\mathcal{O}$.*

In summary, colimits provide a way of combining ontologies by providing a 'superlanguage' in which related ontologies can be embedded. The next section shows that similar support for interoperability can be provided for schemata, and gives a formal relation between ontologies and schemata.

3 Relating Ontologies and Database Schemata

We turn now to conceptual modeling by relational schemata. We give a formal account of schemata and relate these to ontologies. Our main contribution is a cocompleteness result that supports interoperability through merging related schemata.

Definition 3. *A* **schema** *consists of:*

- *a partially ordered set E of* **entity class names***;*
- *a typed collection A of* **attribute names***, i.e., sets $A_{e,s}$ for $e \in E$ and $s \in S$ such that $A_{e,s'} \subseteq A_{e',s}$ whenever $e \leq e'$ and $s \leq s'$;*
- *a partially ordered set R of* **relationship names***;*
- *a typed collection B of* **role names** *for relationships, i.e., sets $B_{r,w}$ for $r \in R$ and $w \in E+D$ (i.e., roles can also be data-valued attributes of relationships), such that $R_{r,w'} \subseteq R_{r',w}$ whenever $r \leq r'$ and $w \leq w'$;*
- *for each class $e \in E$ a set $\mathsf{keys}(e)$, giving* **candidate keys***, which are sets of attributes of e, including a designated primary key denoted $\mathsf{pk}(e)$;*
- *a set Fk of* **foreign keys***, which are triples (k, e_1, e_2) with $e_1, e_2 \in E$, and k a set of attributes of e_1 which is the primary key of e_2; we write Fk_{e_1,e_2} for the set of foreign keys in Fk of the form (k, e_1, e_2) for some k; and*
- *a set F of functional dependencies $\beta \to_e \alpha$, indicating that for $e \in E$, the value of the attribute α depends on the value of the attributes β.*

We denote such a schema by $(E, A, R, B, \mathsf{keys}, Fk, F)$, and often we will just write S or S' for a schema and refer to its components as E or E', A or A', etc.

This definition is based on the Enhanced Entity-Relation model (see, e.g., [4]), and errs on the side of simplicity. For example, we omit weak entities and cardinalities of relationships, although it would be fairly straightforward to include these. Similarly, while all attributes of a class $e \in E$ are required to take values in data types, it is a simple matter to extend the definition with 'entity-valued' attributes; this is allowed for the relationship classes R, for which we also allow data-valued attributes (our use of the term 'role' for the attributes of relationships is not standard, but we adopt it in order to distinguish between attributes of entities and of relationships). We also do not provide for integrity constraints: the only 'axioms' allowed by our definition are functional dependencies. Again, it would be fairly straightforward to allow more expressive constraints by allowing, for example, arbitrary axioms as we do for ontologies.

Example 4. Consider the ontology of Example 2. An entity-relation diagram based on this ontology is shown in Figure 1 below. A schema corresponding to the diagram has entity classes for cars, models, motor-makers and dealers, i.e., the set of class names is $E = \{\mathsf{Car}, \mathsf{Model}, \mathsf{MotorMaker}, \mathsf{Dealer}\}$. The class $\mathsf{MotorMaker}$ has a single attribute name taking values in the datatype $d = \{\mathit{maker1}, \mathit{maker2}\}$, so $A_{\mathsf{MotorMaker},d} = \{\mathsf{name}\}$, and $A_{\mathsf{MotorMaker},d'} = \emptyset$ for all other data types d'.

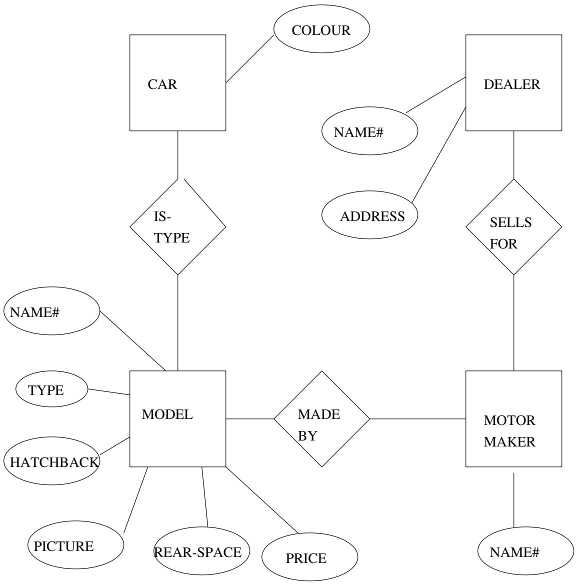

Fig. 1. A diagramatic schema.

There are three relation types, corresponding to the diamond-shaped boxes in the diagram, i.e., $R = \{\mathsf{IsType}, \mathsf{MadeBy}, \mathsf{SellsFor}\}$, each of which has two roles; for example, $B_{\mathsf{SellsFor},\mathsf{Dealer}} = \{\mathsf{dealer}\}$ and $B_{\mathsf{SellsFor},\mathsf{MotorMaker}} = \{\mathsf{maker}\}$, with $B_{\mathsf{SellsFor},w} = \emptyset$ for all other w. Keys and functional dependencies would depend on some conception of how these entities and relations were to be stored; but $\{\mathsf{name}\}$ would presumably be a key for `Model`.

One notion of semantics for such schemata is much the same as that of models for ontologies: the semantics gives sets for entities and relationships, and functions for attributes and roles, and is intended to capture the semantics of actual databases that implement a schema:

Definition 4. *A* **realisation** ρ *of a schema* S *consists of:*

- *sets* ρ_e *for* $e \in E$ *such that* $\rho_e \subseteq \rho_{e'}$ *whenever* $e \leq e'$*;*
- *functions* $\rho_\alpha : \rho_e \to D_d$ *for each attribute name* $\alpha \in A_{e,d}$ *(where* D_d *is the set of appropriate data values for the data type* d*);*
- *sets* ρ_r *for* $r \in R$ *such that* $\rho_r \subseteq \rho_{r'}$ *whenever* $r \leq r'$*;*
- *functions* $\rho_\alpha : \rho_r \to \rho_w$ *for each role name* $\alpha \in R_{r,w}$*;*
- *functions* $\rho_k : \rho_{e_1} \to \rho_{e_2}$ *for each foreign key* $(k, e_1, e_2) \in Fk$*.*

In addition, we require that:

- *keys uniquely identify elements of* ρ_e*:*

$$x = y \quad \text{if} \quad \rho_k(x) = \rho_k(y)$$

for all $x, y \in \rho_e$ *and all* $k \in \mathsf{keys}(e)$*, where the condition in the above equation is an abbreviation for* $\rho_{\alpha_1}(x) = \rho_{\alpha_1}(y)$ *and* $\ldots$ *and* $\rho_{\alpha_n}(x) = \rho_{\alpha_n}(y)$*, where* $\mathsf{keys}(e) = \{\alpha_1, \ldots, \alpha_n\}$*; and*

– *functional dependencies are observed:*

$$\rho_\alpha(x) = \rho_\alpha(y) \quad \textit{if} \quad \rho_\beta(x) = \rho_\beta(y)$$

for all $x, y \in \rho_e$*, whenever the functional dependency* $\beta \to_e \alpha$ *is in* F *(and using the same abbreviation as in the previous equation in case* β *is a set of attributes).*

There are standard techniques for translating a conceptual schema to a relational database schema (see, e.g., [4]). It would be possible to formalise such relational schemata and their realisations in the spirit of Definitions 3 and 4, and then show that the realisations of a relational schema are realisations of the original conceptual schema. We do not do so here, however, prefering instead to concentrate on the relationships between conceptual schemata and ontologies; in particular, we obtain a broader view of the realisations of a schema by relating schemata to ontologies:

Definition 5. *Given a schema* S*, let* $\mathcal{O}_S = (D, \mathsf{C}, A^S, Ax)$ *be the ontology constructed as follows:*

– *the classes are (the disjoint union of) the entities and the relationships, i.e.,* $C = E + R$*;*
– *the attributes are the attributes, roles and foreign keys of* S*, i.e.,*

$$A^S{}_{v,w} = \begin{cases} A_{v,w} & \textit{if } v \in E \textit{ and } w \textit{ is a data type} \\ B_{v,w} & \textit{if } v \in R \\ Fk_{v,w} & \textit{if } v, w \in E \\ \emptyset & \textit{otherwise;} \end{cases}$$

– *the axioms are the key and functional dependency axioms; i.e., for every* $e \in E$ *with primary key* $\mathsf{pk}(e) = \alpha$*,* Ax *contains an axiom of the form*

$$X = Y \textbf{ if } \alpha(X) = \alpha(Y) \ ,$$

where X *and* Y *are variables ranging over the class* e*, and for every functional dependency* $\beta \to_e \alpha$ *in* F*,* Ax *contains an axiom of the form*

$$\alpha(X) = \alpha(Y) \textbf{ if } \beta(X) = \beta(Y) \ ,$$

where X *and* Y *range over the class* e*.*

The relationship between schemata and ontologies is very straightforward; essentially, both specify classes of entities with attributes, together with some axioms constraining their behaviour. The major difference is that a schema makes a distinction between entities and relationships, and also specifies keys that uniquely identify entities. In other words, a schema is an ontology with a little extra structure consisting of keys and the division of classes into entities and relationships. The directness of this relationship is reflected with a correspondingly straightforward relationship between their semantics, stated in Proposition 2 below. However, the directness of the relationship is by no means trivial: if the recipe given in Definition 5 is applied to the schema of Example 4, the resulting ontology is different from the ontology of Example 2 on which the schema was based. The reason for this is that, as the diagram in Example 4 makes clear, new relationship classes have been introduced in moving from the ontology to the schema. A further difference is that the ontology $\mathcal{O}_S$ does not have the axioms of the ontology of Example 2. We discuss these important issues in the final section.

Proposition 2. *There is a one-to-one correspondence between realisations of S and models of $\mathcal{O}_S$.*

Note that this does not say that ontologies and schemata are the same thing; the translation $S \mapsto \mathcal{O}_S$ gives an ontology by 'forgeting' the extra structure of a schema. While it is possible to move in the other direction, i.e., to construct a schema from an ontology, any recipe for doing so involves artificial concoctions for the extra structure. For example, a key could be created for a class that consists of all the attributes of that class, or might be added as a new attribute that gives a unique identifier for members of the class. The very artificiality of such resorts suggests that moving from an ontology to a schema is part of a design process best left to the informed designer.

What the above proposition *does* say is that the translation $S \mapsto \mathcal{O}_S$ strongly preserves the semantics of the schema. In Section 2, we saw that morphisms between ontologies played an essential role in sharing and combining ontologies. We begin our investigation of interoperability of schemata with an appropriate definition of morphism:

Definition 6. *A* **schema morphism** *(or* **view***) $\mu : S \to S'$ is a tuple $\mu = (fA, gA, fB, gB)$, where*

- *$fA : E \to E'$ is a monotone map of partial orders;*
- *gA is a family of maps $gA_{e,d} : A_{e,d} \to A'_{fA(e),d}$ for $e \in E$ and data type d, taking attributes to attributes;*
- *$fB : R \to R'$ is a monotone map of partial orders;*
- *gB is a family $gB_{r,w} : B_{r,w} \to B'_{fB(r),fB(w)}$ for $r \in R$ and $w \in E + D$ (where $fB(d) = d$ for data types d), taking roles to roles.*

We also require that $(gA(k), fA(e_1), fA(e_2)) \in Fk'$ whenever $(k, e_1, e_2) \in Fk$ (i.e., foreign keys are preserved), and that for every $k \in \mathsf{keys}(e)$, there is some $k' \in \mathsf{keys}'(fA(e))$ with $gA(k) \subseteq k'$ (i.e., every key in S forms part of a key in S').

Example 5. Suppose we have a schema S with a class c, which has five attributes α, β, γ, δ and ϵ, with primary key $\{\alpha, \beta\}$, and suppose that γ functionally depends on β, and ϵ on δ. A second normal form for this schema, call it S_2, will have classes d_1 and d_2, where d_1 has attributes α, β, δ and ϵ, with primary key

$\{\alpha, \beta\}$, and where d_2 has attributes β and γ, with primary key $\{\beta\}$. Moreover there is a foreign key (which we denote $f\beta$) from d_1 to d_2. The view from the original schema to S_2 is characterised as follows:

- fA maps c to d_1;
- gA maps α to α, etc., but maps γ to $f\beta;\gamma$, where $f\beta;\gamma$ is a *compound attribute* as in Example 3.

Starting again with the schema S, we could eliminate the non-key dependency $\delta \to \beta$ by factoring c into classes e_1 (with attributes α, β, γ and δ) and e_2 (with attributes δ and ϵ). Call this schema S_3. Again, there is a view $S \to S_3$, the details of which we leave as an exercise for the reader.

As this example suggests, views express the sort of 'refinement' relationships between schemata that are captured by ontology morphisms. In other words, the view of the above example expresses the semantic correctness of reduction to second normal form. However, the notion of correctness here is much weaker than in the case of ontologies, as we explain in the concluding section.

We can now state a major result of our approach: schemata can be 'merged' across common sub-schemata:

Theorem 2. *The category of schemata and views is cocomplete.*

As with Theorem 1, the intuition behind this is that any collection of schemata, given explicit descriptions of their shared concepts, can be merged in a way that preserves their shared components. Roughly speaking, this merging corresponds to integrating schemata by taking their union while preserving commonalities.

Example 6. Consider the schema S and its related schemata S_2 and S_3 from Example 5; the views from S to these latter schemata express the 'lossless decomposition' of eliminating, respectively, partial-key and non-key functional dependencies. The schema S_3, however, is not in 3rd Normal Form, as it is not in 2nd Normal Form (S_2 on the other hand, is in 2nd Normal Form). The colimit given by Theorem 2 gives a (the obvious) schema that is in 3rd Normal Form, by merging the two refinements to remove both functional dependencies.

4 Conclusions

We have given formal definitions of ontology and schema, described how the two concepts are related, and shown that both enjoy 'cocompleteness' properties that support reuse and interoperability. We believe the definitions of schema and realisation are new, and will prove useful in providing a formal basis for the notions of correctness that must be a cornerstone to any body of work on interoperability. While the results of the present paper are just the beginnings of such a body of work, we suggest they represent promising beginnings.

Interoperability of schemata and sharing of knowledge in ontologies both arise naturally in our approach as cocompleteness properties, and their naturality resides in the algebraic nature of our approach. Algebra is, after all, the mathematics of (re)combination, and the results presented here bring algebraic techniques to bear on the important topics of knowledge sharing and interoperability (similar in spirit to recent work by Schorlemmer et al., which places more emphasis on logical morphisms [5, 6]) and suggest avenues of further research that should be very rewarding. For example, an 'amalgamation' property for schemata along the lines of Proposition 1 would get to the heart of operational solutions to interoperating complex databases (Such a property should not be too hard to prove; a similar property is given in an abstract logical setting in [7]). It should be stressed however, that the body of work adumbrated here is still speculative, and there are important technical issues to be addressed.

The most pressing is the nature of the relationship between ontologies and schemata. Proposition 2 gives a very elegant relationship, but the preceding comments suggest that it is not the relationship required in practice. If we begin with an ontology O, and use this to construct a schema S, as in Example 4, then the ontology $\mathcal{O}_S$ differs from O in having both *more* structure (more classes arising from the relationships), and *less* structure (the axioms of O are lost). This describes exactly the situation of ontologies being related: O and $\mathcal{O}_S$ are related through their 'common denominator' — an ontology without the axioms of O and without the extra classes of $\mathcal{O}_S$. Our future studies of the semantical bases for interoperability will centre on how colimits of schemata and ontologies interact with such relations, and in particular on how these figure in the design of integration schemata.

References

1. Berners-Lee, T., Hendler, J., O.Lassila: The semantic web. Scientific American **279** (2001)
2. Bench-Capon, T., Malcolm, G., Shave, M.: Semantics for interoperability: relating ontologies and schemata. Technical Report ULCS-02-029, Dept. of Computer Science, Univ. Liverpool (2002)
3. Bench-Capon, T., Malcolm, G.: Formalising ontologies and their relations. In Bench-Capon, T., Soda, G., Tjoa, A.M., eds.: Proc. 10th International Conf. on Database and Expert Systems Applications (DEXA'99), Springer Lecture Notes in Computer Science volume 1677 (1999) 250–259
4. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 2nd edn. Addison-Wesley (1994)
5. Schorlemmer, M.: Duality in knowledge sharing. Technical Report EDI-INF-RR-0134, Division of Informatics, Univ. Edinburgh (2002)
6. Kalfoglou, Y., Schorlemmer, M.: Information-flow-based ontology mapping. Technical Report EDI-INF-RR-0135, Division of Informatics, Univ. Edinburgh (2002)
7. Alagić, S., Bernstein, P.A.: A model theory for generic schema management. In Grahne, G., Ghelli, G., eds.: DBLP 2001. Volume 2397 of Springer Lecture Notes in Computer Science. (2002) 228–246