Semantics for Interoperability: relating ontologies and schemata

Trevor Bench-Capon
tbc@csc.liv.ac.uk

Grant Malcolm
grant@csc.liv.ac.uk

Michael Shave
mshave@csc.liv.ac.uk

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

Abstract

Any builder of an information system, whether a database or a knowledge based system, will start from some conceptualisation of the domain, which will embody a number of fundamental assumptions about the domain. Often these underlying assumptions will remain implicit. If we wish to share information across different information systems, it is important to be aware of these assumptions.

In knowledge based systems these assumptions are often made explicit by producing an ontology, an explicit formalisation of the conceptualisation of the domain. Such ontologies are used to facilitate sharing and reuse between two systems, and formulate rules which will map between divergent but compatible conceptualisations.

These underlying assumptions are equally pertinent if we want to allow to databases to interoperate. Recent work has suggested that the information contained in their schemata is insufficient, and that the schemata need to be enriched by metadata which makes available these assumptions. Some of this work has even made explicit mention of the use of an ontology to represent the commonality of the conceptualisations within which the databases were developed.

Several stages are required to move from a conceptualisation to a design. We must select the relevant information, decide how it is to be stored, how it is to be accessed, etc. Different levels of schema can be seen as milestones along the route of this transformation. Alternatively, given two or more databases, we can see the recovery of their underlying conceptualisation (ontology) as an important step towards integrating them into an interoperating system.

Understanding this requires that we can be precise about what constitutes a compatible conceptualisation, what characterises a schema as opposed to an ontology, and how schema relate to ontologies.

Elsewhere we have developed a formalism for ontologies which allows us to answer such questions with respect to ontologies. In this paper we present a compatible formalism for database schemata, which allows us to characterise precisely the relationships between schemata and an ontologies, and give some foundations for understanding how these concepts can be used to facilitate interoperability of heterogeneous information systems.

1 Introduction

In constructing any information system, one must inevitably start from some perception of the the domain which one is attempting to model. This view will stem in part from the background knowledge and previous experience of the builder, and it will be enhanced or refined by discussions with those familiar with the domain and by the reading of relevant documentation. However there is a great temptation for both the builder and the domain experts to regard some information as 'too obvious to mention'. These implicit assumptions then result in a system which has unsubstantiated additions or omissions, each of which may or may not be justifiable.

An information system built on an uncertain and incompletely defined basis such as this is like a house built on sandy foundations: both are liable to collapse if their structure is put under unexpected pressure. In the case of the information system, the pressure is represented by unforeseen queries which do not share the implicit assumptions of the system designer, and which consequently fail, or draw incorrect conclusions from the data which they use.

Fortunately, as experience and understanding grows, it is possible to discern a steady movement away from ad hoc designs and towards generalisation, the identification of concepts, and generic approaches to the design and definition of information systems.

This trend can be clearly seen in the evolution of database design. Traditional application-based files were superseded first by multi-task files as a rudimentary form of general-access database, and then by software based on the famous three-level architecture of a database advocated by ANSI/SPARC. Codd's formulation of the Relational model in 1970 [10] then gave a tremendous boost to the concept of a schema as an application-independent definition of the structure and content of a database. In its turn, the simplistic format of a strict relational database has been generalised to Non-First Normal Form models and the increasingly common Object Oriented approach.

The existence of a schema for a database has also been instrumental in providing a basis for interoperation between databases. The definition of a global schema for a distributed database, or an integration schema for a federated database, has been an essential step in the creation of these important forms of information system. The latter, in particular, depending as it does on the recognition of areas where there is a common or compatible conceptualisation of data, also offers some useful guidance for the more general problem of interoperation between two knowledge bases (KBSs).

Just as the organised representation of data led to the concept of a schema to convey the metadata of a database, there has been increasing recognition of the corresponding need for a vehicle to express the metadata of a KBS. There has thus been widespread, though often fuzzy, reference to the concept of an ontology for a domain or a KBS. Without such a framework it is in fact difficult, if not impossible, to characterise two KBSs, or to state whether they are in some sense compatible. The latter issue is of profound importance in view of the widespread interest in sharing or merging knowledge [34, 36, 1, 24]. This interest stems from the added value which is potentially available when it is possible to take advantage of the fresh perspectives provided by interoperation between two or more independent but compatible knowledge sources.

1.1 The nature of an ontology

Just as a schema is the formal structure which underpins a DB, an ontology is the formal structure which underpins a KBS (or, indeed, any collection of knowledge). There is still much debate about the precise definition of an ontology, but it is important to make clear from the outset that in this paper we regard an ontology, just as much as a schema, as a formal statement. It is unfortunate that it is common practice to refer loosely to 'the ontology' of a situation, a task, or an event - meaning thereby an informal understanding of the component objects and operations which we associate with some domain. For example we might refer to the ontology of the social security system, and have in mind concepts such as claimant, contribution-record, and pensionable age. However these are ideas which merely constitute what we could describe as the conceptualisation of the domain. To create an ontology of the domain we must produce formal definitions of each of these relevant ideas, and usually much else besides.

The significance of the distinction between a formally-defined ontology and the informal understanding of an underlying conceptualisation is well expressed by Guarino [25]:

> It is important to stress that an ontology is language-dependent, while a conceptualisation is language-independent . In its de facto use in AI, the term 'ontology' collapses the two aspects, but a clear separation between them becomes essential to address the issues related to ontology sharing, fusion and translation, which in general imply multiple vocabularies and multiple conceptualisations.

An ontology, just like a database schema, is defined in relation to some context. It provides a basis for knowledge sharing within that context by providing a common terminology and frame of reference, whereas the role of a schema is the creation of a well defined data structure on which data transformations can be expressed. An ontology is not concerned with procedures which are built on the ontological concepts, nor with the implementational details of the generic operations which it defines. An ontology identifies concepts, but it does not in general identify instances of those concepts. Keys, in the database sense, have no part to play in an ontology because keys are required essentially for data retrieval or modification, and these tasks are not relevant to the definition of an ontology.

1.2 Synergy between schemata and ontologies

We can thus see an ontology as providing the basic terminology, conceptual definitions and generic constraints for an information system, while a schema using that ontology specifies the operational rules and dynamic constraints which define the manner in which in which data can be accessed and updated.

Nevertheless these two points of view are intrinsically linked. Just as one cannot discuss functions without their operands, and entities are worthless without functions which act on them, there is a similar synergy between ontologies and schemata. One would not take the trouble to define the formal ontology of a domain unless that domain was to be the subject of knowledge processing operations. Conversely, the description of operations on a domain inevitably requires understanding of the concepts which underly that domain.

In current practice there is often a spectrum rather than a clear cut division between the ontological definition of a knowledge source and its rule-based or schema specification. Some ontological description languages such as CLASSIC [6] or LOOM [28] permit the inclusion of a significant amount of operational detail in the definition of an ontology. Conversely, an object-oriented approach encourages the production of schemata which take a concept-based view of a domain rather than the traditional task-based enterprise model.

In order to merge knowledge from a number of information systems, it is necessary first of all to identify a common conceptual level where all the systems have a common interpretation of knowledge, drawn from a shared conceptualisation of that level. Recent work has shown that an explicit statement of this conceptionalisation in terms of an ontology, or Common Concept Model [37], provides a formal basis on which the remaining stages establishing interoperability can be grounded [32, 31]. Once this has been done, attention can turn from the conceptual to the procedural aspects of interoperation, which will typically require the definition of a suitable schema. To emphasise its relationship with the original information systems, we refer to this as an integration schema. It should be noted that one shared ontology can be the basis for more than one integration schema if different aspects of knowledge are to be shared - for example, there may be an integration schema for financial knowledge and another for personnel information.

The distinction between a shared ontology and an integration schema has sometimes been blurred, but the argument above shows that they are neither conflicting nor identical. Instead, they are two distinct and complementary aspects of interoperation, both of which must be the subject of formal definitions, and both of which are essential in all but the very simplest cases. The former is the more fundamental, the latter the more directly related to operational issues. To implement the interoperation of heterogeneous information systems, both aspects will need to be addressed, and there is a spectrum of choices about their relative importance.

However, the nature of the relationship between a shared ontology and an associated integration schema remains, at present, intuitive. We must find a way to express this relationship more precisely and more formally if we hope to construct information systems with the ability to share knowledge which are robust and well-defined. In a recent paper [3], Bench-Capon and Malcolm tackled similar issues in relation to the specification of ontologies, by setting out an abstract and formal methodology in which ontologies are presented as algebraic theories whose semantics are given by classes of models. In this paper we extend and generalise this notation to the specification of schemata, and thereby provide a basis on which the relationship between a shared ontology and an associated integration schema can be expressed in formal and unambiguous terms.

In section 2 we introduce our formal description of ontologies, in section 3 we show how this can be used to describe interesting relations between ontologies, in section 4 we provide a compatible description of schemata, and show how this can be used to describe interesting relations between schemata and between schemata and ontologies, and in section 5 we make some concluding remarks.

2 Ontological Specifications

Semantics is of primary importance in designing any language, and a language for formally specifying ontologies should have a clear and precise semantics. In particular, relationships between ontologies are best described in terms of their semantics. We take an algebraic approach, in which ontologies are presented as algebraic theories whose semantics is given by classes of models. This allows us to characterise relations between ontologies as relations between their classes of models. In this section we present an abstract syntax for specifying ontologies and develop the model theory that provides semantics for these specifications. We are much less interested in presenting a concrete syntax; the motivating examples given below are in an *ad hoc* syntax.

The presentation of ontologies below follows [3], and is rather dense, assuming some familiarity with algebraic specifications (introductions can be found in [33, 12]) and also with basic category theory (see [27, 2] for introductions).

Our ontologies specifiy classes of entities with attributes. These attributes take values in data types such as numbers, booleans, lists and so on. Sometimes it is convenient to make changes in the types of attributes, for example an ontology may be refined by refining the types of some of its attributes, so we consider these types to be a part of ontological specifications. We formalise this part of specifications using the notion of *order-sorted algebraic theory*; [21, 17] give details of order-sorted algebra, the following is an example of a theory for the natural numbers, in the notation of the language OBJ [23, 17].

```
th NAT is
  sorts Nat NonZeroNat .
  subsort NonZeroNat < Nat .
  op 0 : -> Nat .
  op s : Nat -> NonZeroNat .
  op p : NonZeroNat -> Nat .
```

```
    var N : Nat .
    eq  p(s(N)) = N .
  endth
```

The details of this notation are not essential to the present paper. The main point is that an order-sorted theory presents some sorts, partially ordered by a subsort relation, some typed operations, and some axioms (here just one equation). Models of such specifications interpret sorts as sets, and operations as functions (of the appropriate type); these sets and functions should satisfy the axioms in the obvious sense.

The data types in our ontologies are given by an order-sorted theory together with a fixed model interpreting its sorts and operations:

Definition 1 A **data domain** is a pair (T, D), where T is an order-sorted equational theory $T = (\mathsf{S}, \Sigma, E)$, where $\mathsf{S} = (S, \leq)$ is the partially ordered set of sort names, Σ is the collection of typed operation symbols, and E is the set of equations, and where D is a model of T. We often write D instead of (T, D). □

We do not require that D be an initial model of T [22], although in many cases that would be an obvious choice. Any computable algebra can be specified equationally [4]; even uncomputable algebras, such as the reals, can be used in ontological specifications by fixing an appropriate model D (for example, the reals provide a model of the theory `NAT`). An important advantage of using order-sorted algebra is that it is implemented in languages such as OBJ [23, 17] and CafeOBJ [11]. This provides basic machine support for prototyping and theorem proving.

A *theory morphism* $\phi : T \to T'$ between order-sorted theories induces a functor from models of T' to models of T (see, e.g., [16], though Proposition 7 presents the essential ideas). This means that any model D' of T' gives rise to a model $\phi D'$ of T; this allows us to define morphisms of data domains as follows:

Definition 2 A **morphism of data domains** $\psi : (T, D) \to (T', D')$ is a pair $\psi = (\phi, h)$, where $\phi : T \to T'$ is an order-sorted theory morphism and $h : \phi D' \to D$ is a T-homomorphism. □

Order-sorted theory morphisms (see [21]) are pairs (f, g), where f is a monotonic map from sort names to sort names, and g maps operation symbols to operation symbols.

Data morphisms go from coarse to fine structures. For example, one ontology might specify an attribute with values in a data type shade, interpreted in one domain as $\{\mathsf{light}, \mathsf{dark}\}$, while another ontology may specify values of this attribute in the range $[0, 100)$. A domain morphism might then translate all values up to 50 as being light, and all values between 50 and 100 as being dark. The mapping

$$i \mapsto \begin{cases} \mathsf{light} & \text{if } 0 \leq i < 50 \\ \mathsf{dark} & \text{if } 50 \leq i < 100 \end{cases}$$

describes the homomorphism part of the above definition (i.e., we have h mapping i in $\phi D'_{\mathsf{shade}} = (0, 100]$ to $D_{\mathsf{shade}} = \{\mathsf{light}, \mathsf{dark}\}$).

Definition 3 An **ontology signature**, or just **signature** for short, is a triple $(\mathcal{D}, \mathsf{C}, A)$, where $\mathcal{D} = (T, D)$ is a data domain, $\mathsf{C} = (C, \leq)$ 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 $e \in C + S$, where S is the set of sorts in the order-sorted theory T. 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 $\mathsf{S} + \mathsf{C}$. □

For $c \in C$ and $s \in S$, the set $A_{c,s}$ represents attributes of the class c that take values in D_s. Attributes may also take values in classes: for $c, c' \in C$, the set $A_{c,c'}$ represents attributes of the class c that take values in the class c'. The final condition of the above definition states that if $c \leq c'$ then the class c inherits all attributes of the class c'.

Example 4 The following is an ontology for different kinds of cars that might be used by a second-hand car dealer. We use an *ad hoc* notation that hopefully requires little explanation. 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. We omit the data domain from the specification, but all names beginning with a `*` represent data types, and the notation `{white,red,blue}` represents a data type with three elements `white`, `red`, and `blue`.

```
Car
  colour: {white,red,blue}
  model: Model
  year:  *year
  price: *pounds

Estate-Car < Car
  rear-space:  *square-metres

Saloon-Car < Car
  hatchback: (y n)

Model
  name: *string
  manufacturer: {maker1, maker2}
  photo: *gif
```

The class `Car` has four attributes, e.g., `model`, which takes values in the class `Model` (i.e., `model` $\in A_{\mathtt{Car},\mathtt{Model}}$). The class `Estate-Car` has five attributes: four inherited from `Car` plus the attribute `rear-space`, which takes values in a data type `*square-meters` (as we said, we omit the data domain from this specification, but it would be sensible for this data type to consist of some kind of numerical values, just as the type `*gif` of the `photo` attribute should represent some graphics encoding). □

We turn now to the semantics of such specifications:

Definition 5 A **model** M of an ontology signature $(\mathcal{D}, \mathsf{C}, A)$ consists of:

- A C-sorted family of sets M_c for $c \in C$, called the **carriers** of M;
- for each attribute symbol $\alpha \in A_{c,e}$, a function $M_{c:\alpha} : M_c \to M_e$, where if $e \in S$ then $M_e = D_e$, subject to the following monotonicity requirement: for $c \le c' \in C$ and $\alpha \in A_{c',e}$, we have $M_{c':\alpha}\restriction_{M_c} = M_{c:\alpha}$.

Because of the above monotonicity property, we write simply M_α instead of $M_{c:\alpha}$.

Given models M and N of (C, A), a **homomorphism of models** $h : M \to N$ is a C-sorted family of functions $h_c : M_c \to N_c$ such that for all $\alpha \in A_{c,e}$, $h_c; N_\alpha = M_\alpha; h_e$, where $h_e = 1_{D_e}$ if $e \in S$. □

An ontology consists of a signature and some axioms; its denotation is the class of all models that satisfy the axioms. Before we give the formal definitions, we consider morphisms that relate ontologies:

Definition 6 An **ontology signature morphism** $\chi : (\mathcal{D}, \mathsf{C}, A) \to (\mathcal{D}', \mathsf{C}', A')$ is a tuple $\chi = (\phi, f, g)$, where $\phi : \mathcal{D} \to \mathcal{D}'$ is a data domain morphism, $f : \mathsf{C} \to \mathsf{C}'$ 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 $f(e) = \phi(e)$ if $e \in S$ (i.e., if $e \in S$ then $g_{c,e}$ maps attributes that take values in D_e to attributes that take values in $(\phi D')_e$), such that if $c \le c'$ and $e \le e'$ then $g_{c',e}(\alpha) = g_{c,e'}(\alpha)$ for all $\alpha \in A_{c',e}$. □

Just as with order-sorted theories, signature morphisms give rise to functors on models:

Proposition 7 A signature morphism $\chi : (\mathcal{D}, \mathcal{A}) \to (\mathcal{D}', \mathcal{A}')$ induces a functor, which we also call χ, from the category of $(\mathcal{D}', \mathcal{A}')$-models to the category of $(\mathcal{D}, \mathcal{A})$-models. The functor is defined on objects by

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

where M is a model of $(\mathcal{D}', \mathcal{A}')$. □

The final component in an ontology specification consists of axioms constraining the possible values of attributes. In this paper we consider axioms to be just conditional equations, but other kinds of sentences, such as horn clauses, are also possible.

Definition 8 An **ontology** is a pair (Σ, Ax) where Σ is a signature and Ax a set of axioms. A **model** of such an ontology is a model M of Σ that satisfies each axiom in Ax, in which case we write $M \models Ax$.

An **ontology morphism** $\chi : (\Sigma, Ax) \to (\Sigma', Ax')$ is a signature morphism $\chi : \Sigma \to \Sigma'$ such that $\chi M \models Ax$ whenever $M \models Ax'$. □

Example 9 As an example, we give an ontology for models of cars that might be used for a buyers' guide:

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

Model
 name: *string
 type: {estate, saloon}
 hatchback: {y, n, n/a}
 picture: *gif
 rear-space: *square-metres
 manufacturer: Motor-maker
 price: *pounds

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

Dealer
  name: *string
  address: *string


Axioms:
  var M : Model
  hatchback(M) = n/a  if  type(M) = estate
  rear-space(M) = 0   if  type(M) = saloon
```

The axioms in this ontology specify particular values of attributes `hatchback` and `rear-space` depending on the kind of `model`. □

3 Relating Ontologies

Ontology morphisms provide a direct way of relating ontologies; any morphism $\chi : \mathcal{O} \to \mathcal{O}'$ states that the ontology $\mathcal{O}'$ is a *refinement* of the ontology $\mathcal{O}$. This is because by definition, any model of $\mathcal{O}'$ is (or more precisely, gives rise to, by Proposition 7 and Definition 8) a model of $\mathcal{O}$. This approach is quite standard in algebraic specification (see, e.g., [35]). If the morphism χ is an *inclusion* morphism, i.e., the various components of χ are all inclusions (of class names, attribute names, etc.), so that $\mathcal{O}'$ is really just an extension of $\mathcal{O}$, then χ as a functor from models of $\mathcal{O}'$ to models of $\mathcal{O}$ is a forgetful functor, fitting standard intuitions about refinements.

A more general way of relating ontologies is through pairs of morphisms:

Definition 10 A **relation** between ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ consists of an ontology $\mathcal{O}$ and a pair of morphisms $\chi_i : \mathcal{O} \to \mathcal{O}_i$ for $i = 1, 2$. □

A morphism $\chi : \mathcal{O}_1 \to \mathcal{O}_2$ is a special case of a relation, by taking $\mathcal{O} = \mathcal{O}_1$ and $\chi_1 = 1_{\mathcal{O}}$.

3.1 Combining and Sharing Ontologies

We view ontologies as representing some knowledge of a given domain from a specific point of view. Thus, Examples 4 and 9 specify various models of cars, from the points of view of second-hand car dealers and buyers' guides, repectively. In order to support knowledge sharing, it is important that ontologies can be merged. The following theorem (from [3]) states that any group of related ontologies can be merged:

Theorem 11 The category of ontologies and their morphisms is cocomplete. □

This is stated in the language of category theory [27], and says that there is a 'best' way (a *colimit*) of combining related ontologies that share some subcomponents. Example 12 below illustrates the construction of colimits. Again, colimits are standard in combining algebraic theories (for example [7]; see also [26], whose cocompleteness results for order-sorted theories is used to obtain a colimiting data domain as part of the proof of Theorem 11).

The following shows how this result allows our two car ontologies to be merged across a common subcomponent.

Example 12 Consider the following ontology, which provides a common subcomponent for the ontologies of Examples 4 and 9.

```
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 4. 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 9, 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` is 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 values in `{maker1,maker2}` as desired.

In the colimit of this relation (i.e., the pushout), there is just one attribute `photo`, representing 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, which tells us that 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
  year:  *year
  price: *pounds

Estate-Car < Car
  rear-space:  *square-metres

Saloon-Car < Car
  hatchback: (y n)

Model
 name: *string
 type: {estate, saloon}
 hatchback: {y, n, n/a}
 picture: *gif
 rear-space: *square-metres
 manufacturer: {maker1, maker2}
 manufacturer2: Motor-maker
 price: *pounds

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

Dealer
  name: *string
  address: *string


Axioms:
  var M : Model
  manufacturer(M) = name(manufacturer2(M))
  hatchback(M) = n/a  if  type(M) = estate
  rear-space(M) = 0   if  type(M) = saloon
```

Note that ontologies can be related in different ways, and the ontology $\mathcal{O}$ does not really correspond to the 'intersection' of two ontologies, but is created in such a way as to state just what is to be shared between ontologies. For example, we might have omitted the `manufacturer` attribute from $\mathcal{O}$, which would correspond to a judgement that the two attributes of that name in $\mathcal{O}_1$ and $\mathcal{O}_2$ were distinct. We might have included an attribute `price` in $\mathcal{O}$, mapped to the attribute of the same name in $\mathcal{O}_1$ and to the compound attribute `model;price` in $\mathcal{O}_2$. This would express the judgement that these two attributes correspond to the same feature. However, in this example, we have not done this because these features are in fact distinct; for the second-hand car dealer, the price is an individual car's saleable value, and for a buyers' guide, the price is the model's new price. $\Box$

3.2 Compatibility of Ontologies

Our definition of model requires data domains to be interpreted in a fixed way. This means that not all ontologies have models, because the axioms in an ontology might not be satisfiable. That is, the axioms might be inconsistent in the sense that they entail *true* = *false*, while the fixed data model in the data domain requires that these values be distinct. We say that an ontology is **consistent** iff it has at least one model (so that the axioms are satisfiable). Now we can give a notion of compatibility for ontologies.

Definition 13 Let $\chi_i : \mathcal{O} \to \mathcal{O}_i$ for $i = 1, 2$ be a relation between ontologies. We say that $\mathcal{O}_1$ and $\mathcal{O}_2$ are **compatible (over $\mathcal{O}$)** iff their colimit is consistent. $\Box$

We have the following 'amalgamation' property (from [3]):

Proposition 14 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 $\mathcal{O}$. $\Box$

Essentially, this says that a Grothendieck category of ontologies and their models is cocomplete; more intuitively, if models of $\mathcal{O}_1$ and $\mathcal{O}_2$ agree on their shared parts (i.e., on $\mathcal{O}$), then those models can be 'merged' to provide a model of the colimiting ontology. Similar results in an algebraic setting can be found in [30, 8].

4 Relating Ontologies and Database Schemata

In this section, we give a formal definition of conceptual schema, along the lines of our definition of ontology in Section 2. As with ontologies, our schemata have a fixed data domain, providing representations of the data types used for the values of attributes. Again, the motivation for this is that data domains give data types for which it is useful to have some fixed representation. Conceptual schemata build on this by specifying classes of entities and relationships that are abstract in the sense that no particular representation is required; the responsibility for finding suitable representations lies with those who implement an actual database from a schema.

Definition 15 A **schema** consists of:

- a data domain D;
- 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 $s \in S$, where S is the set of sort names from the data domain, 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 $w \in E{+}S$ (note that what we call 'roles' also include 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 we take to be 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

$$(D, E, A, R, B, \mathsf{keys}, Fk, F) \ ,$$

and often we will just write S or S' for a schema and refer to its components as D or D', E or E', etc. $\Box$

This definition is based on the Enhanced Entity-Relation model (see, e.g., [13]), and errs on the side of simplicity[1]. 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

[1] Even though a definition with eight components should not be described as 'simple'!

'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 (these axioms might be equational, Horn Clause, or even full first-order predicate sentences). We also deviate from standard practice in giving an explicit presentation of foreign keys; this simplifies our treatment of views (Definition 21 below).

Example 16 Consider the ontology of Example 9. An entity-relation diagram based on this ontology is shown in Figure 1 below. A schema correspond-

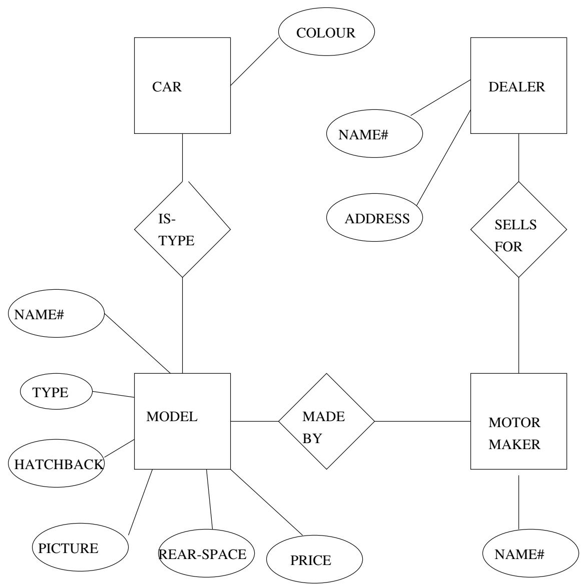

Figure 1: A diagramatic schema.

ing to the diagram has entity classes for cars, models, motor-makers and dealers, i.e., the set of class names is $E = \{\texttt{Car}, \texttt{Model}, \texttt{MotorMaker}, \texttt{Dealer}\}$. The class `MotorMaker` has a single attribute `name` taking values in the datatype $s = \{\texttt{maker1}, \texttt{maker2}\}$, so $A_{\texttt{MotorMaker},s} = \{\texttt{name}\}$, and $A_{\texttt{MotorMaker},s'} = \emptyset$ for all other data types s'. There are three relation types, corresponding to the diamond-shaped boxes in the diagram, i.e., $R = \{\texttt{IsType}, \texttt{MadeBy}, \texttt{SellsFor}\}$, each of which has two roles; for example, $B_{\texttt{SellsFor},\texttt{Dealer}} = \{\texttt{dealer}\}$ and $B_{\texttt{SellsFor},\texttt{MotorMaker}} = \{\texttt{maker}\}$, with $B_{\texttt{SellsFor},w} = \emptyset$ for all other w. Keys and functional dependencies would depend on some conception of how these entities and relations were to be stored; but $\{\texttt{name}\}$ 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 17 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 $\rho_\alpha : \rho_e \to D_s$ for each attribute name $\alpha \in A_{e,s}$ (subject to the monotonicity constraint of Definition 5);
- 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 $\alpha \in R_{r,w}$ (again, subject to the monotonicity constraint of Definition 5);
- functions $\rho_k : \rho_{e_1} \to \rho_{e_2}$ for each foreign key $(k, e_1, e_2) \in \mathit{Fk}$.

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* ... *and* $\rho_{\alpha_n}(x) = \rho_{\alpha_n}(y)$, where $\mathsf{keys}(e) = \{\alpha_1, \ldots, \alpha_n\}$; and
- functional dependencies are observed:
$$\rho_\alpha(x) = \rho_\alpha(y) \quad \text{if} \quad \rho_\beta(x) = \rho_\beta(y)$$
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., [13]). It is possible to formalise such relational schemata and their realisations in the spirit of Definitions 15 and 17, 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 18 Given a schema S, let $\mathcal{O}_S = (D, \mathsf{C}, A^S, Ax)$ be the ontology constructed as follows:

- the data domain is the same, i.e., D;

- 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.,

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

- 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 $\beta \rightarrow_e \alpha$ 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 Propositions 19 and 20 below. However, the directness of the relationship is by no means trivial: if the recipe given in Definition 18 is applied to the schema of Example 16, the resulting ontology is different from the ontology of Example 9 on which the schema was based. The reason for this is that, as the diagram in Example 16 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 9. We discuss these important issues in the final section.

Proposition 19 Every realisation ρ of a schema S gives a model $\mathcal{O}_\rho$ of the ontology $\mathcal{O}_S$.

Proof. For $c \in C = E + R$, define $(\mathcal{O}_\rho)_c = \rho_c$ (noting that this picks out ρ's interpretation of entities or of relationships, depending on whether c is in E or in R); similarly, for any attribute α in A^S, we take $(\mathcal{O}_\rho)_\alpha = \rho_\alpha$, whether α be an attribute of an entity or a role of a relationship. Finally, with these interpretations it is immediate from Definition 17 that $\mathcal{O}_\rho$ satisfies the axioms Ax of $\mathcal{O}_S$. □

In fact, we have the stronger result:

Proposition 20 There is a one-to-one correspondence between realisations of S and models of $\mathcal{O}_S$.

Proof. Let M be a model of $\mathcal{O}_S$; we construct a realisation $\mathcal{R}_M$ of S as follows. For $e \in E$, let $(\mathcal{R}_M)_e = M_e$, and for any attribute α in A, let $(\mathcal{R}_M)_\alpha = M_\alpha$. Relationships and roles are treated in the same way, and it follows from Definition 18 that $\mathcal{R}_M$ satisfies the key and functional dependency constraints. Finally, it is clear that the mapping $M \mapsto \mathcal{R}_M$ is inverse to the mapping $\rho \mapsto \mathcal{O}_\rho$ of Proposition 19. □

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

- $\phi : D \to D'$ is a data domain morphism;
- $fA : E \to E'$ is a monotone map of partial orders;
- gA is a family of maps $gA_{e,s} : A_{e,s} \to A'_{fA(e),\phi(s)}$ for $e \in E$ and $s \in S$, 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 $w \in E + S$ (where $fB(w) = \phi(w)$ if $w \in S$), 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 22 Suppose we have a schema with a class e, which has four attributes α, β, γ and δ (we are not interested here in their result types), and suppose the primary key of e is $\{\alpha, \beta\}$, and that the attribute δ functionally depends on β. The second normal form for this schema will have classes e_1 and e_2, where e_1 has attributes α, β and γ, with primary key $\{\alpha, \beta\}$, and where e_2 has attributes β and δ, with primary key $\{\beta\}$. Moreover there is a foreign key (β) from e_1 to e_2. The view from the original schema to its second normal form is characterised as follows (using the same order as the bullet-points of definition 21):

- ϕ is just the identity, as both use the same data domain;
- fA maps e to e_1;
- gA maps α to α, etc., but maps δ to $f\beta;\delta$, where $f\beta$ is the foreign key from e_1 to e_2, and $f\beta;\delta$ is a *compound attribute* as in Example 12;
- fB and gB are empty, as we do not consider relations in this example.

$\square$

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 reuction 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 subschemata:

Theorem 23 The category of schemata and views is cocomplete.

Sketch of Proof. The general construction of colimits is very similar to that of ontologies [3], so we merely sketch here the construction of pushouts. Let $v_i : \mathcal{S} \to \mathcal{S}_i$ be views for $i = 1, 2$, and assume for simplicity that all data domains are the same; a pushout can be constructed as follows. The set of entity class names is given by the pushout of $fA_i : E \to E_i$ in the category of partial orders; this is a standard construction (see e.g., [5]), and gives a partially ordered set whose elements can be seen as equivalence classes of $E_1 + E_2$. If the pushout cocone is $fA'_i : E_i \to E'$ for $i = 1, 2$, then E' gives the entity classes of the colimiting schema S', and the maps fA'_i give the corresponding components of the views to the colimit.

Let $e' \in E'$, and for a fixed data type s construct the diagram whose edges consist of all morphisms $A_{e,s} \to (A_i)_{fA_i(e),s}$ whenever $fA_i(e) \in e'$ (where e' is viewed as an equivalence class of $E_1 + E_2$); the set ${A'}_{e',s}$ is constructed as a colimiting object for this diagram. The family of sets of attribute names, A', is defined by constructing these sets for each $e' \in E'$ and data type s. The injections of these cocones give the components gA' of the views to S'.

Relations and roles are treated in essentially the same way. Keys are constructed as follows: let $e' \in E$ be an equivalence class (as above), and for each $e \in E_i$ $(i = 1, 2)$ with $fA_i(e) \in e'$, pick some $k \in \mathsf{keys}_i(e)$; apply gA' to each attribute in k, denoting the result by $gA'(k)$, and take the union of all these sets (i.e., for each $e \in E_i$ $(i = 1, 2)$ with $fA_i(e) \in e'$), giving one candidate key in $\mathsf{keys}'(e')$; all candidate keys are constructed by repeating this process for each possible choice of $k \in \mathsf{keys}_i(e)$ (when each k is the primary key of its class, the resulting union of sets of attributes is the primary key of e').

Finally, the set of functional dependencies is obtained by translating the functional dependencies in S_1 and S_2 along the maps gA'. $\square$

As with Theorem 11, 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.

5 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 in computer science has led to new approaches to parameterisation and software composition [14, 15, 30]. In particular, the relationships between our formalisation of ontologies and schemata on the one hand and 'hidden algebra' [19, 18, 20] on the other, seem particularly relevant to the axioms arising from keys and functional dependencies: these can be viewed as 'hidden congruences', i.e., structure-preserving relations betwen states of concurrent abstract machines; the importance of these in system composition is explored in [18, 29, 9].

The results presented here bring algebraic techniques to bear on the important topics of knowledge sharing and interoperability, and suggest avenues of further research that should be very rewarding. For example, an 'amalgamation' property for schemata along the lines of Proposition 14 would get to the heart of operational (i.e., at the level of what we call 'realisations') solutions to interoperating complex databases. 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. Propositions 19 and 20 give a very elegant relationship, but the comments preceding Proposition 19 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 16, 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 (Definition 10): 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] Yigal Arens, Craig A. Knoblock, and Chun-Nan Hsu. Query processing in the sims information mediator. In Austin Tate, editor, *Advanced Planning Technology*. AAAI Press, Menlo Park, CA, 1996.

[2] Michael Barr and Charles Wells. *Category Theory for Computing Science*. Prentice Hall, 1990.

[3] Trevor Bench-Capon and Grant Malcolm. Formalising ontologies and their relations. Submitted for publication, 1999.

[4] Jan A. Bergstra and John V. Tucker. Algebraic specifications of computable and semicomputable data types. *Theoretical Computer Science*, 50:137–181, 1987.

[5] Francis Borceux. *Handbook of Categorical Algebra*, volume 1. Cambridge University Press, 1994.

[6] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67. ACM Press, 1989.

[7] Rod Burstall and Joseph A. Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *Proceedings, 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292–332. Springer, 1980. Lecture Notes in Computer Science, Volume 86.

[8] Corina Cîrstea. A semantical study of the object paradigm. Transfer thesis, Oxford University Computing Laboratory, 1996.

[9] Corina Cîrstea. Coalgebra semantics for hidden algebra: parameterized objects and inheritance. In F. Parisi-Presicce, editor, *Proc. 12th Workshop on Algebraic Development Techniques*. Springer-Verlag Lecture Notes in Computer Science, 1376, 1998.

[10] E.F. Codd. A relational model of data for large, shared data banks. *Communications of the ACM*, 13:377–387, 1970.

[11] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[12] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer, 1985.

[13] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 2nd edition, 1994.

[14] Joseph A. Goguen. Principles of parameterized programming. In Ted Biggerstaff and Alan Perlis, editors, *Software Reusability, Volume I: Concepts and Models*, pages 159–225. Addison Wesley, 1989.

[15] Joseph A. Goguen. Hyperprogramming: A formal approach to software environments. In *Proceedings, Symposium on Formal Approaches to Software Environment Technology*. Joint System Development Corporation, Tokyo, Japan, 1990.

[16] Joseph A. Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, 1992.

[17] Joseph A. Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. MIT Press, 1996.

[18] Joseph A. Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, 1999. To appear.

[19] Joseph A. Goguen and Grant Malcolm. Hidden coinduction: behavioral correctness proofs for objects. *Mathematical Structures in Computer Science*, 1999. To appear.

[20] Joseph A. Goguen, Grant Malcolm, and Tom Kemp. A hidden Herbrand theorem. In Catuscia Palamidessi, Hugh Glaser, and Karl Meinke, editors, *Principles of Declarative Programming*, pages 445–4622. Springer-Verlag Lecture Notes in Computer Science 1490, 1998.

[21] Joseph A. Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992.

[22] Joseph A. Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM T.J. Watson Research Center, October 1976. In *Current Trends in Programming Methodology, IV*, Raymond Yeh, editor, Prentice-Hall, 1978, pages 80–149.

[23] Joseph A. Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph A. Goguen and Grant Malcolm, editors, *Software Engineering with OBJ: Algebraic Specification in Practice*. to appear. Also available as a technical report from SRI International.

[24] P.M.D. Gray, A. Preece, N.J. Fiddian, W.A. Gray, T.J.M Bench-Capon, M.J.R. Shave, N. Azarmi, M. Wiegand, M. Ashwell, M. Beer, Z. Cui, B. Diaz, S.M. Embury, K. Hui, A.C. Jones, D.M. Jones, G.J.L. Kemp, E.W. Lawson, K. Lunn, P. Marti, J. Shao, and Visser P.R.S. Kraft: Knowledge fusion from distributed databases and knowledge bases. In *Eighth International Workshop on Database and Expert Systems Applications*, pages 682–691. IEEE Press: Los Alamitos, 1997.

[25] N. Guarino. Formal ontology and information systems. In *Proceedings, 1st International Conference on Formal Ontologies in Information Systems (FOIS'98)*, pages 3–15, 1998.

[26] Anne E. Haxthausen and Friederike Nickl. Pushouts of order-sorted algebraic specifications. In *AMAST '96*. Springer-Verlag Lecture Notes in Computer Science 1101, 1996.

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

[28] R.M. MacGregor. A deductive pattern matcher. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 403–408. AAAI Press, 1988.

[29] Grant Malcolm. Behavioural equivalence, bisimilarity, and minimal realisation. In Magne Haveraaen, Olaf Owe, and Ole-Johan Dahl, editors, *Recent Trends in Data Type Specifications. 11th Workshop on Specification of Abstract Data Types, WADT11. Oslo Norway, September 1995*, pages 359–378. Springer-Verlag Lecture Notes in Computer Science 1130, 1996.

[30] Grant Malcolm. Interconnection of object specifications. In Stephen Goldsack and Stuart Kent, editors, *Formal Methods and Object Technology*. Springer Workshops in Computing, 1996.

[31] Nayyer Masood and Barry Eaglestone. Semantics bases schema analysis. In *Proceedings of DEXA98*, pages 80–89. Springer Verlag Lecture Notes in Computer Science 1460, 1998.

[32] Nayyer Massod. *Semantics for Schema Analysis*. PhD thesis, University of Bradford, 1999.

[33] Karl Meinke and John V. Tucker. Universal algebra. In S. Abramsky, D. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 189–411. Oxford University Press, 1993.

[34] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senatir, and W. Swartout. Enabling technology for knowledge sharing. *AI Magazine*, 12(3):36–56, 1991.

[35] Donald Sannella and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications. *Acta Informatica*, 25:233–281, 1988.

[36] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.

[37] C. Yu, W. Sun, S. Dao, and D. Keirsey. Determining relationships among attributes for interoperability of multi-database systems. In *Proceedings of the 1st International Workshop on Interoperability in Multi-Database Systems*, pages 251–7. IMS Kyoto, 1991.