Theoretical Aspects of Schema Merging *

P. Buneman, S. Davidson and A. Kosky
Department of Computer and Information Sciences
University of Pennsylvania
Philadelphia, PA 19104-6389

January 24, 1994

Abstract

A general technique for merging database schemas is developed that has a number of advantages over existing techniques, the most important of which is that schemas are placed in a partial order that has bounded joins. This means that the merging operation, when it succeeds, is both associative and commutative, *i.e.*, that the merge of schemas is independent of the order in which they are considered — a property not possessed by existing methods. The operation is appropriate for the design of interactive programs as it allows user assertions about relationships between nodes in the schemas to be considered as elementary schemas. These can be combined with existing schemas using precisely the same merging operation.

The technique is general and can be applied to a variety of data models. It can also deal with certain cardinality constraints that arise through the imposition of keys. A prototype implementation, together with a graphical interface, has been developed.

*This research was supported in part by ARO DAAL03-89-C-0031PRIME and NSF IRI 8610617, Peter Buneman was also supported by a UK SERC funded visit to Imperial College, London.

1 Introduction

The desire to provide user views that combine existing databases, and to combine user views during the design process of new databases, leads directly to the problem of schema merging — a problem that has been present in the database literature for at least ten years and to which a variety of sophisticated techniques have been applied. At one end of the spectrum, the user is provided with a set of tools for manipulating two schemas into some form of consistency [1, 2]; at the other end, algorithms have been developed that take two schemas, together with some constraints, and create a merged schema [3]. In general, one will want to use a method that lies somewhere between these two extremes; a number of such variations have been explored, and are surveyed in [4]. It appears that some user manipulation of the given schemas is essential — especially to introduce consistent names — but that a merging algorithm can also be very useful, especially when large schemas are involved.

To the best of our knowledge, the question of what meaning or semantics this merging process should have has not been explored. Indeed, several of the techniques that have been developed are *heuristics*: there is no independent characterization of what result they should produce. One would like to have some semantic basis for a merge that would characterize it with some notion of consistency with the associated data. This semantic basis should be related to the notion of an instance of a schema, and is discussed in [5]. In this paper, we shall develop a simple and general characterization of database schemas that allows us to give natural definitions of what a merge is in terms of the informational content of the schemas being merged. In particular we shall define a merge which takes the union of all the information stored in a collection of database schemas, and, when possible, forms a schema presenting this but no additional information. We shall be able to define a binary merging operator that is both commutative and associative, which means that if two or more schemas are merged, the result is independent of the order in which the merges are performed. Existing techniques do not have this property. Worse still, lack of associativity is endemic to the data models against which the merging process is commonly defined, such as the Entity-Relationship (ER) model.

Using a more general formalism, we will be able to rescue this situation by introducing special, additional information during the merging process. The additional information describes its own origin, and can be readily identified to allow subsequent merges to take place. In addition,

our new schema merging technique may be applied to other, existing models, such as the ER-model, by first translating the schemas of these models into our model, then carrying out the merge, and finally translating back into the original model. It is possible to show that, if such an approach is used, then the merging process respects the original model.

The paper is organized as follows: we shall first describe a general data model that subsumes, in some sense, data models such as relational, entity-relationship and functional. We then observe that even in this general model, we cannot expect to have associative merges. We relax the constraints on our model to produce a still more general formulation of a *weak* schema for which the merging process is well behaved, and then show how to convert these weak schemas back into our original schemas. We also show that certain common forms of constraints on the schemas, such as key constraints and some cardinality constraints, can be handled in the same framework. Finally we shall indicate how these methods could be used equally well to give definitions of alternative merge, and, in particular, describe how we could form *lower merges* representing the intersection of the information represented by a collection of schemas.

2 The Model

We represent a schema as a directed graph, subject to certain restrictions, whose nodes are taken from a set $\mathcal{N}$ of *classes*, and with two kinds of edges that are used to represent "attribute of" relationships or "specialization of" relationships between classes. Attribute edges have labels taken from a set $\mathcal{L}$, so that we represent the attribute-of edges by a relation $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{L} \times \mathcal{N}$. If $(p, a, q) \in \mathcal{E}$ then we write $p \xrightarrow{a} q$, with the meaning that any instance of the class p must have an a-attribute which is a member of the class q. Since in some data models, like the ER data model, the term "attribute" is used to designate a certain kind of node in a schema, we shall use the neutral term *arrow* to refer to these labeled relationships, and say for $p \xrightarrow{a} q$ that *p has an a-arrow to class q*. The specialization edges are represented by a relation $\mathcal{S}$ on classes; we use the notation $p \Longrightarrow q$ and say that p is a *specialization* of q when $(p, q) \in \mathcal{S}$. This indicates that all the instances of p are also instances of q. Formally, a schema over $\mathcal{N}$, $\mathcal{L}$ is a triple $(\mathcal{C}, \mathcal{E}, \mathcal{S})$ where $\mathcal{C} \subseteq \mathcal{N}$ is a finite set of classes, $\mathcal{S}$ is a partial order (a reflexive transitive and antisymmetric relation on $\mathcal{C}$), and $\mathcal{E}$ is a subset of $\mathcal{C} \times \mathcal{L} \times \mathcal{C}$

satisfying

1. If $p \xrightarrow{a} q_1$ and $p \xrightarrow{a} q_2$ then $\exists s \in \mathcal{C}$. $s \Longrightarrow q_1$ and $s \Longrightarrow q_2$ and $p \xrightarrow{a} s$.
2. If $p \Longrightarrow q$ and $q \xrightarrow{a} r$ then $p \xrightarrow{a} r$.
3. If $p \xrightarrow{a} s$ and $s \Longrightarrow r$ then $p \xrightarrow{a} r$.

for all $a \in \mathcal{L}$ and $p, q, r, s \in \mathcal{C}$

The first constraint says that if p has an a-arrow, then there is *least* class s (under the ordering $\mathcal{S}$) such that p has an a-arrow to class s. Such a class is said to be the **canonical** class of the a-arrow of p. The second constraint says that, if q has an a-arrow to class r and p is a specialization of q, then p must also have an a-arrow to class r. The third constraint says that, if p has an a-arrow to class s and s is a specialization of r, then p also has an a-arrow to class r, so constraints 2 and 3 together mean that arrows are, in some sense, preserved by specialization. It is worth remarking that we could equally well have defined the arrows as partial functions from classes to classes, which is how they are expressed in the definition of a functional schema in [2]. If we write $p \xrightarrow{a} q$ when p has an a-arrow with canonical class q, we have the equivalent conditions

D1. If $p \xrightarrow{a} q_1$ and $p \xrightarrow{a} q_2$ then $q_1 = q_2$

D2. If $q \xrightarrow{a} s$ and $p \Longrightarrow q$ then $\exists r \in \mathcal{C}. r \Longrightarrow s$ and $p \xrightarrow{a} r$

Also, given any $\rightharpoonup$ satisfying conditions D1 and D2, if we define the relation $\longrightarrow$ by $p \xrightarrow{a} q$ iff there exists a $s \in \mathcal{C}$ such that $s \Longrightarrow q$ and $p \xrightarrow{a} s$, then $\longrightarrow$ will satisfy conditions 1, 2 and 3 above. Conditions D1 and D2 are those given for the arrow in [2] and are also given by Motro [1] as axioms for functional schemas (the latter uses unlabeled arrows).

For example, the ER diagram shown in figure 1 corresponds to the database schema shown in figure 2, where single arrows are used to indicate edges in $\mathcal{E}$ and double arrows are used to represent pairs in $\mathcal{S}$ (double arrows implied by the transitivity and reflexivity of $\mathcal{S}$ are omitted). In all the subsequent diagrams, edges in $\mathcal{E}$ implied by constraint 2 above, will also be omitted.

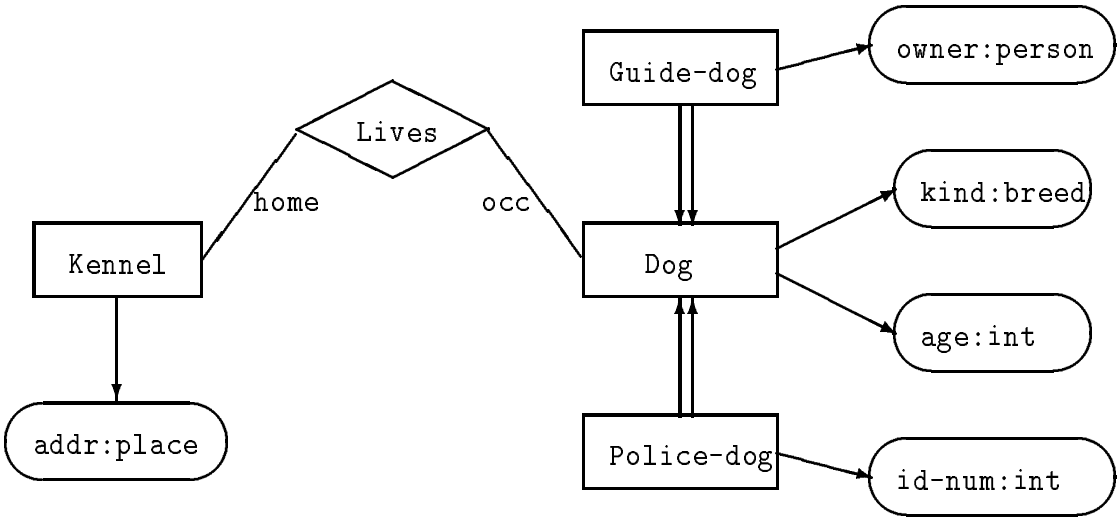

Figure 1: An ER-diagram with "isa" relations

Suitable restrictions of such graphs may be used to describe instances of a variety of data models: relational, entity-relationship and functional. For a relational instance, we stratify $\mathcal{N}$ into two classes $\mathcal{N}_R$ and $\mathcal{N}_A$ (relations and attribute domains), disallow specialization edges, and restrict arrows to run labeled with the name of the attribute from $\mathcal{N}_R$ to $\mathcal{N}_A$ (first normal form), while, for the E-R model, we stratify $\mathcal{C}$ into three classes (attribute domains, entities and relationships) and again place certain restrictions on the edges. Moreover, it can be shown that the merging process described in section 4 preserves these restrictions, so that we can merge schemas from other models by first translating them into our model, then merging them, and finally translating them back into the original data model (see [5] for details). By a less constrained process we can describe instances of the functional model [6, 2, 1]. The graphs are also general enough to represent databases with higher order relations (that is, relationships between relationships), and complex data structures (such as circular definitions of entities and relationships), features that are commonly found in object-oriented data models. Consequently, despite its apparent simplicity, the generality of the model makes it a good candidate for schema merging. One should note, however, that further adornment of these graphs is needed to describe instances of sophisticated data models such as those proposed in [7] and [8], which contain constructors for sets and variants.

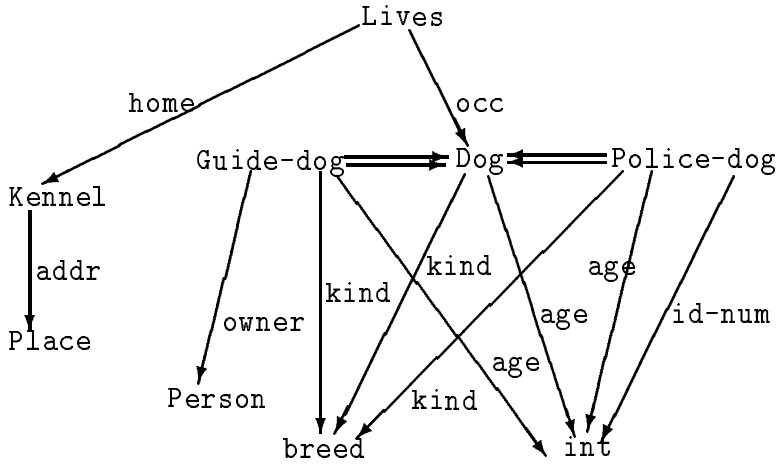

Figure 2: A database schema with "isa" relations

3 Problems with finding merges of schemas

The first problem to be resolved when forming a common merged schema for a number of distinct databases is to state the correspondences between the classes and correspondences between the arrow labels of the various databases. This problem is inherently *ad hoc* in nature, and depends on the real-world interpretations of the underlying databases. Therefore, the designer of the system must be called upon to resolve naming conflicts, whether homonyms or synonyms, by renaming classes and arrows where appropriate. The interpretation that the merging process places on names is that if two classes in different schemas have the same name, then they are the same class, regardless of the fact that they may have different arrow edges. For example, if one schema has a class `Dog` with arrow edges `License#`, `Owner` and `Breed`, and another schema has a class `Dog` with arrow edges `Name`, `Age` and `Breed`, then the merging process will collapse them into one class with name `Dog` and arrow edges `License#`, `Owner`, `Name`, `Age`, and `Breed`. It is also possible to constrain the merging process by introducing specialization relations $a_1 \Longrightarrow a_2$ between nodes a_1 in schema $\mathcal{G}_1$ and a_2 in schema $\mathcal{G}_2$. We can treat $a_1 \Longrightarrow a_2$ as an atomic schema that is to be merged with $\mathcal{G}_1$ and then with $\mathcal{G}_2$.

Because our schema merge is associative and commutative, the result is well-defined; indeed an arbitrary set of constraints can be added in this fashion.

For the remainder of this section and the following section, we will consider the merge of a collection of schemas to be a schema that presents all the information of the schemas being merged, but no additional information (although in Section 6, we will indicate that there may be other, equally valid interpretations of what the merge should be). Hence what we will consider to be the merge is the "least upper bound" of the database schemas under some sort of information ordering.

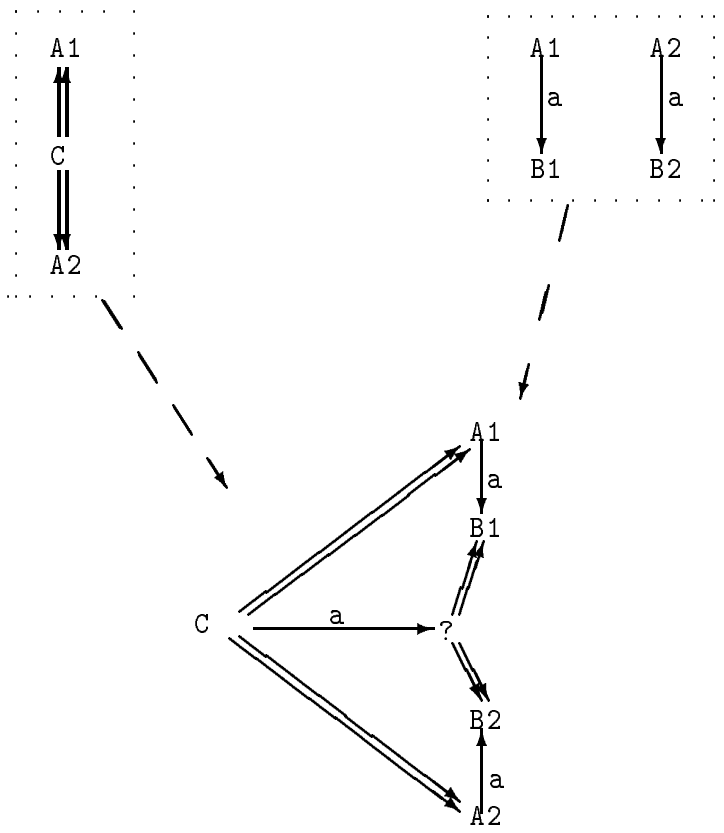

Figure 3: Schema merging involving implicit classes

One of the first problems we notice in looking for a suitable definition of merge is that

the merge of two schemas may contain extra *implicit* classes in addition to the classes of the schemas being merged. For example, figure 3 shows two schemas being merged. The first schema asserts that the class `C` is a subclass of both the classes `A1` and `A2`. The second schema asserts that the classes `A1` and `A2` both have `a`-arrows, of classes `B1` and `B2` respectively. Combining this information, as we must when forming the merge, we conclude that `C` must also have an `a`-arrow, and that this arrow must be of both the class `B1` and `B2`. Consequently, due to the restrictions in our definition of database schemas in Section 2, the `a`-arrow from the class `C` must point to class which is a specialization of both `B1` and `B2` and so we must introduce such a class into our merged schema.

When we consider these "implicit" classes further we find that it is not sufficient merely to introduce extra classes into a schema with arbitrary names: the implicit classes must be treated differently from normal classes. Firstly, if we were to give them the same status as ordinary classes we would find that binary merges are not associative.

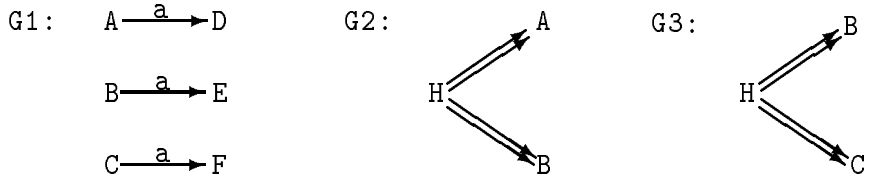

Figure 4: Some simple schemas

For example consider the three simple schemas shown in figure 4. If we were to first merge the schemas `G1` and `G2` we would need to introduce a new implicit class (`X?`) below `D` and `E`, and then merging with `G3` would make us introduce another new class below `X?` and `F`, yielding the first schema shown in figure 5. On the other hand, if we were to merge `G1` with `G3` and then merge the result with `G2`, we would first introduce an implicit class below `E` and `F`, and then introduce another one below this one and `D`. Clearly what we really want is *one* implicit class which is a specialization of all three of `D`, `E` and `F`.

Another problem is that it is possible for one schema to present more information than another without containing as many implicit classes. Intuitively, for one schema to present all the information of another (plus additional information) it must have, at least, all the normal classes of the other. However let us consider the two schemas shown in figure 6. We would

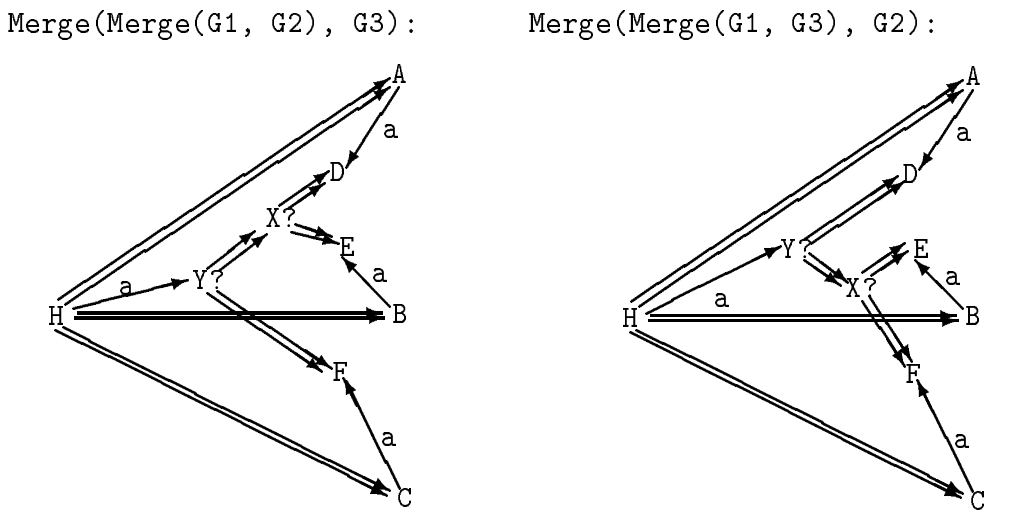

Figure 5: An example of non-associative merging

like to assert that the schema G3 shown in figure 7 is the merge of the two schemas, G1 and G2, but the schema G4 also presents all the information of G1 and G2, and in addition contains fewer classes than G3. The point is that G4 asserts that the a-arrow of F has class E, which may have restrictions on it in addition to those which state that it is a subclass of both C and D, while G3 only states that the a-arrow of F has both classes C and D.

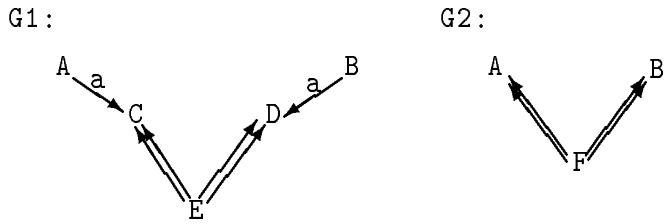

Figure 6: Yet more schemas

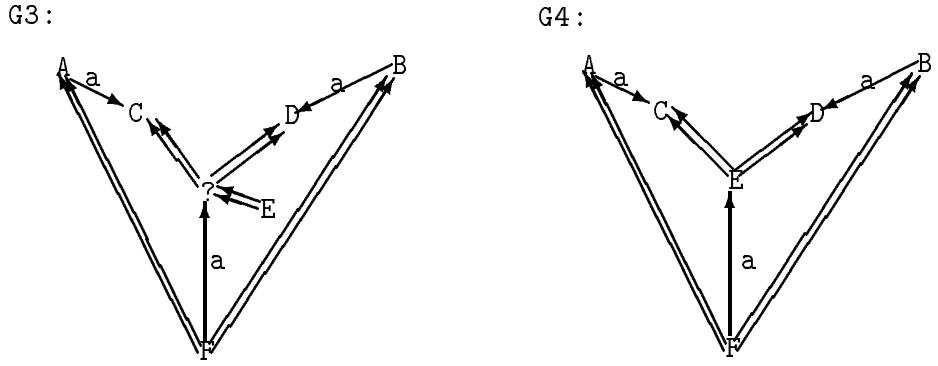

Figure 7: Possible candidates for the merges of the schemas

4 Merging Database Schemas

In order to avoid the complexities of introducing implicit classes, we will weaken our definition of database schemas so that implicit classes become unnecessary. We then define an information ordering on these *weak schemas*, such that binary joins do exist and are associative, and form the weak schema merge. Finally we convert the merged weak schema into a proper schema by introducing additional implicit classes (we will refer to the schemas satisfying the conditions in section 2 as *proper schemas.*)

4.1 Weak Schemas

A **weak schema** is a schema in which we no longer require that if class p has an a-arrow, then the a-arrow has a canonical class (condition 1 of proper schemas). Formally, a weak schema over $\mathcal{N}$, $\mathcal{L}$ is a triple $(\mathcal{C}, \mathcal{E}, \mathcal{S})$ where $\mathcal{C} \subseteq \mathcal{N}$ is a set of classes, $\mathcal{S}$ is a partial order (a reflexive transitive and antisymmetric relation on $\mathcal{C}$), and $\mathcal{E}$ is a subset of $\mathcal{C} \times \mathcal{L} \times \mathcal{C}$ satisfying

W1. If $p \Longrightarrow q$ and $q \stackrel{a}{\longrightarrow} r$ then $p \stackrel{a}{\longrightarrow} r$.

W2. If $p \stackrel{a}{\longrightarrow} s$ and $s \Longrightarrow r$ then $p \stackrel{a}{\longrightarrow} r$.

for all $a \in \mathcal{L}$ and $p, q, r, s \in \mathcal{C}$

The ordering on weak schemas is defined in the obvious way: Given two weak schemas $\mathcal{G}_1 = (\mathcal{C}_1, \mathcal{E}_1, \mathcal{S}_1)$ and $\mathcal{G}_2 = (\mathcal{C}_2, \mathcal{E}_2, \mathcal{S}_2)$, we write $\mathcal{G}_1 \sqsubseteq \mathcal{G}_2$ iff

1. $\mathcal{C}_1 \subseteq \mathcal{C}_2$
2. $\mathcal{E}_1 \subseteq \mathcal{E}_2$
3. $\mathcal{S}_1 \subseteq \mathcal{S}_2$

That is, every class in $\mathcal{G}_1$ appears in $\mathcal{G}_2$, every a-arrow edge in $\mathcal{G}_1$ appears in $\mathcal{G}_2$, and every specialization edge in $\mathcal{G}_1$ appears in $\mathcal{G}_2$.

It is clear that $\sqsubseteq$ is a partial ordering on weak schemas; it is also bounded complete, as shown in the following proposition.

Proposition 4.1 *For any weak schemas $\mathcal{G}_1$ and $\mathcal{G}_2$, if there exists a weak schema $\mathcal{G}'$ such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}'$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}'$ then there is a least such weak schema $\mathcal{G}_1 \sqcup \mathcal{G}_2$.*

Proof: Given weak schemas $\mathcal{G}_1$ and $\mathcal{G}_2$ as above, define $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S})$ by

$$\begin{aligned}
\mathcal{C} &= \mathcal{C}_1 \cup \mathcal{C}_2 \\
\mathcal{S} &= (\mathcal{S}_1 \cup \mathcal{S}_2)^* \\
\mathcal{E} &= \{p \xrightarrow{a} s \in (\mathcal{C} \times \mathcal{L} \times \mathcal{C}) \mid \exists q, r \in \mathcal{C} \cdot p \Longrightarrow q \in \mathcal{S},\ r \Longrightarrow s \in \mathcal{S}, \\
&\qquad\qquad \text{and } q \xrightarrow{a} r \in (\mathcal{E}_1 \cup \mathcal{E}_2)\}
\end{aligned}$$

(where $(\mathcal{S}_1 \cup \mathcal{S}_2)^*$ denotes the transitive closure of $(\mathcal{S}_1 \cup \mathcal{S}_2)$, and $\mathcal{E}$ adds edges to $\mathcal{E}_1 \cup \mathcal{E}_2$ necessary for conditions W1 and W2 to hold). It is clear that, if $\mathcal{G}$ is a weak schema, then it is the least weak schema such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}$. Hence it is sufficient to show that, if there is a weak schema, $\mathcal{G}' = (\mathcal{C}', \mathcal{E}', \mathcal{S}')$, such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}'$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}'$, then $\mathcal{G}$ is indeed a weak schema. The only way that $\mathcal{G}$ can fail to be a weak schema is if the relation $\mathcal{S}$ fails to be antisymmetric, so the result follows from the fact that, for any suitable $\mathcal{G}'$ as above, we must have $\mathcal{S} \subseteq \mathcal{S}'$, and so if $\mathcal{S}'$ is antisymmetric then so is $\mathcal{S}$. ∎

We say a finite collection of weak schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, is **compatible** if the relationship $(\mathcal{S}_1 \cup \ \ldots \ \cup \mathcal{S}_n)^*$ is anti-symmetric (where $\mathcal{S}_1, \ldots, \mathcal{S}_n$ are the specialization relations of

$\mathcal{G}_1, \ldots, \mathcal{G}_n$ respectively). Consequently we have, for any finite compatible collection of proper schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, there exists a *weak schema merge* $\mathcal{G} = \bigsqcup_{i=1}^{n} \mathcal{G}_i$. Furthermore, since we define $\mathcal{G}$ as the least upper bound of $\mathcal{G}_1, \ldots, \mathcal{G}_n$, the operation is associative and commutative.

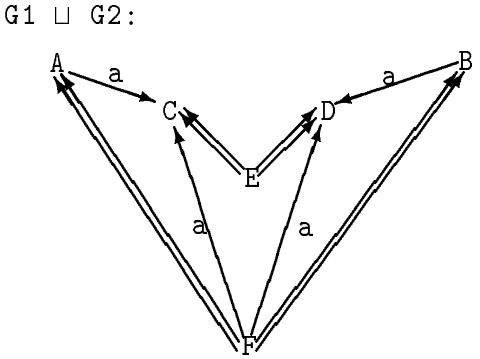

Figure 8: The least upper bound of two schemas

For example, the schemas `G1` and `G2` in figure 6 are compatible, and their weak schema merge is shown in figure 8.

4.2 Building proper schemas from weak schemas

We now must pay the price for our use of weak schemas: we must provide a way of introducing implicit classes into a weak schema $\mathcal{G}$, in order to form a proper schema $\overline{\mathcal{G}}$, such that if there are any proper schemas greater than $\mathcal{G}$ then $\overline{\mathcal{G}}$ is such a schema.

First we introduce some new notation. For any $p \in \mathcal{C}$ and any $a \in \mathcal{L}$, we write $\mathcal{R}(p, a)$ to denote the set of classes reachable from p via a-arrows

$$\mathcal{R}(p, a) = \{q \in \mathcal{C} \mid p \xrightarrow{a} q\}$$

Further, for any set $X \subseteq \mathcal{C}$, we use $\mathcal{R}(X, a)$ to denote the set of classes reachable from classes in X via a-arrows

$$\mathcal{R}(X, a) = \{q \in \mathcal{C} \mid \exists p \in X \cdot p \xrightarrow{a} q\}$$

We define the function $Min_{\mathcal{S}} : \mathcal{P}(\mathcal{N}) \to \mathcal{P}(\mathcal{N})$ so that, for any set $X \subseteq \mathcal{N}$, $Min_{\mathcal{S}}(X)$ is the set of minimal elements of X under the ordering $\mathcal{S}$. That is

$$Min_{\mathcal{S}}(X) = \{p \in X \mid \forall q \in X \cdot \text{if } q \Longrightarrow p \text{ then } q = p\}$$

where $\mathcal{P}(A)$ denotes the power set of the set A.

We now proceed to build a proper schema $\overline{\mathcal{G}} = (\overline{\mathcal{C}}, \overline{\mathcal{E}}, \overline{\mathcal{S}})$ from $\mathcal{G}$ as follows:

1. First we will construct a set, $Imp \subseteq \mathcal{P}(\mathcal{N})$, of sets of classes, corresponding to our implicit classes. We will construct Imp via a series of auxiliary definitions as follows:

$$\begin{aligned} I^0 &= \{\{p\} \mid p \in \mathcal{C}\} \\ I^{n+1} &= \{\mathcal{R}(X, a) \mid X \in I^n, a \in \mathcal{L}\} \\ I^\infty &= \bigcup_{n=1}^{\infty} I^n \\ Imp &= \{Min_{\mathcal{S}}(X) \mid X \in I^\infty \text{ and } |Min_{\mathcal{S}}(X)| > 1\} \end{aligned}$$

   Intuitively, Imp is the set of all sets of minimal classes which one can reach by following a series of arrows from some class in $\mathcal{C}$, with cardinality greater than 1. Note that the process of forming Imp will halt since there are a finite number of subsets of $\mathcal{P}(\mathcal{N})$.

2. We define $\overline{\mathcal{C}}$ by first taking $\mathcal{C}$ and then adding a new class $\overline{X}$ for every $X \in Imp$. That is

$$\overline{\mathcal{C}} = \mathcal{C} \cup \{\overline{X} \mid X \in Imp\}$$

3. We define $\overline{\mathcal{E}}$ so that if $p \xrightarrow{a} q$ for each $q \in X$ then $p \xrightarrow{a} \overline{X} \in \overline{\mathcal{E}}$, while if there is a q such that $p \xrightarrow{a} q$ then $p \xrightarrow{a} q \in \overline{\mathcal{E}}$. Formally:

$$\begin{aligned} \overline{\mathcal{E}} &= \{x \xrightarrow{a} q \mid x \in \overline{\mathcal{C}}, a \in \mathcal{L}, q \in \mathcal{R}(x, a)\} \\ &\quad \cup \{x \xrightarrow{a} \overline{Y} \mid x \in \overline{\mathcal{C}}, a \in \mathcal{L}, Y \in Imp, Y \subseteq \mathcal{R}(x, a)\} \end{aligned}$$

   where $\mathcal{R}(\overline{X}, a) = \mathcal{R}(X, a)$ for all $X \in Imp$.

4. We define $\overline{\mathcal{S}}$ by first taking $\mathcal{S}$ and then adding every $\overline{X} \Longrightarrow \overline{Y}$ such that every class in Y has a specialization in X; every $\overline{X} \Longrightarrow p$ where p has a specialization in X; and every

$p \Longrightarrow \overline{X}$ where p is a specialization of every class in X.

$$\begin{aligned}\overline{\mathcal{S}} \;=\; & \mathcal{S} \cup \{\overline{X} \Longrightarrow \overline{Y} \mid X,Y \in Imp, \forall p \in Y \cdot \exists q \in X \cdot q \Longrightarrow p \in \mathcal{S}\} \\ & \cup \{\overline{X} \Longrightarrow p \mid X \in Imp, p \in \mathcal{C}, \exists q \in X \cdot q \Longrightarrow p \in \mathcal{S}\} \\ & \cup \{p \Longrightarrow \overline{X} \mid p \in \mathcal{C}, X \in Imp, \forall q \in X \cdot p \Longrightarrow q \in \mathcal{S}\}\end{aligned}$$

For example, the effect on the schema shown in figure 8 would be to introduce a single implicit class, $\overline{\{C,D\}}$, thus forming the schema `G3` shown in figure 7 with the class `?` replaced by $\overline{\{C,D\}}$.

It can be shown that for any weak schema $\mathcal{G}$, $\overline{\mathcal{G}}$ is a weak schema and $\mathcal{G} \sqsubseteq \overline{\mathcal{G}}$. Furthermore, $\overline{\mathcal{G}}$ can be shown to respect condition 1 of the definition of a proper schema, and is therefore also a proper schema.

We would like to be able to show that $\overline{\mathcal{G}}$ is the least proper schema greater than $\mathcal{G}$. However there are two minor problems with this: first, it is possible to form other similar proper schemas by using different names for the implicit classes (compare this to alpha-conversion in the lambda calculus); second, for any two sets $X,Y \in Imp$, if every class in Y has a specialization in X then our method will include the pair $(\overline{X},\overline{Y})$ in $\overline{\mathcal{S}}$. However it is not necessarily the case that this specialization relation is required, and it might be safe to omit it. We could attempt to modify our method so that such pairs are only introduced when required. Instead we will argue that, since the implicit classes have no additional information associated with them, it follows that these specialization relations do not introduce any extra information into the database schema, and so, since they seem natural, it is best to leave them there. Consequently we feel justified defining the **merge** of a compatible collection of database schemas, $\mathcal{G}_1,\ldots,\mathcal{G}_n$, to be the database schema $\overline{\mathcal{G}}$, where $\mathcal{G} = \bigsqcup_{i=1}^{n} \mathcal{G}_i$.

Of course, not every merge of a collection of compatible schemas makes sense. That is, the new classes introduced may have no correspondence to anything in the real world. To capture this semantic aspect of our model, we would need to introduce a "consistency relationship" on $\mathcal{N}$, and require that, for every $X \in Imp$ and every $p,q \in X$, the pair (p,q) is in the consistency relationship. If this condition were violated, the schemas would be **inconsistent**, and $\overline{\mathcal{G}}$ would not exist. Note that checking consistency would be very efficient, since it just requires examining the consistency relationship. However, while the idea is interesting, it is beyond the scope of this paper. Suffice it to say that if the merge of $\mathcal{G}_1,\ldots,\mathcal{G}_n$ fails, either

because $\mathcal{G}_1, \ldots, \mathcal{G}_n$ are *incompatible*, or because they are *inconsistent*, the merge should not proceed, and the user must re-assess the assumptions that were made to describe the schemas.

5 Cardinality Constraints and Keys

The model we have used so far concentrates on the semantic relationships between classes via specialization and arrow edges, but does not further describe arrows as participating in keys or having associated cardinality constraints. Cardinality constraints in the ER model are typically indicated on the edges between a relationship and an entity by labeling them "many" (or "N", indicating unrestricted upper bounds), or "1" (sometimes indicated by an arrow pointing into the entity set).[1] For example, consider the `Advisor` relationship between `Faculty` and `GS` in Figure 9, and for the moment interpret this schema as an ER diagram. As drawn, `Advisor` is a "many-many" relationship, typically indicated by labeling the `faculty` and `victim` edges "N": a graduate student may be advised by several faculty members, and each faculty member can advise several different graduate students. If we decided to restrict `Advisor` to indicate that a graduate student can be advised by at most one faculty member, the `faculty` edge from `Advisor` to `Faculty` would be relabeled "1".

As it stands, however, our model has no way of distinguishing these different edge semantics. Using the example of the previous paragraph, labeling the `faculty` edge in the `Advisor` relationship "1" rather than "N" could result in the same graph in our model, *i.e.* the graph in Figure 9.[2] In this section, we will capture such constraints by introducing "key constraints" on nodes, and argue that in some sense they are more general than the cardinality constraints typically found in ER models.

Key constraints, which indicate that certain attributes of an entity form a *key* for that entity, are another common form of assertions found in database models. As an example, in the ER

[1] It is worth noting that there is little agreement on what edge labels to use, and what they mean in ER digrams, especially for ternary and higher degree relationships. No semantics are given in [9]. Introductory textbooks on databases avoid the question and merely give examples of binary relationships [10, 11, 12]; [13] is slightly more honest and says that "the semantics of ternary and higher-order relationship sets can become quite complex to comprehend." Varying interpretations can be found in [14, 15, 3].

[2] Of course, one might eliminate the `Advisor` node entirely, and draw a single `Advisor`-edge from `GS` to `Faculty`, but this reasoning does not extend to ternary and higher degree relationships.

and relational models, for the entity set `Person(SS#, Name, Address)`, we might claim that there are two keys: {`SS#`} and {`Name, Address`}. The intuition behind this statement is that if two people have the same social security number, or the same name and address, then they are the same person. Generalizing, one could claim that a set of *edges* of a relationship form a key for that relationship. As an example, for the `Advisor` relationship in which the `faculty` edge was labelled with a "1" and the `victim` edge was labelled with a "N", we could claim that the `victim` edge forms a key. In the terminology of proper schemas, we capture such key constraints by asserting that $\{a_1, a_2, ..., a_n\}$ form a key for p, where each a_i is the label of some arrow out of p.

In many established data-models it is required that every class has at least one key, so that the set of all the arrows of a class forms a key if no strict subset of the arrows does. By relaxing this constraint, so that a class may have no key at all, we can capture models in which there is a notion of object identity. A *superkey* of a class is any superset of a key. We may therefore think of the set of superkeys for a class p, $\mathcal{SK}(p)$, as a set of sets of labels of arrows out of p. $\mathcal{SK}(p)$ has the property that it is "upward closed", *i.e.* if $s \in \mathcal{SK}(p)$ and $s\prime \supseteq s$, then $s\prime \in \mathcal{SK}(p)$.

We now have the constraint on specialization edges that if $p \Longrightarrow q$ then $\mathcal{SK}(p) \supseteq \mathcal{SK}(q)$, *i.e.* all the keys for q are keys (or superkeys) for p. For example, the specialisation `Advisor` $\Longrightarrow$ `Committee` in the schema shown in figure 9 asserts that the advisor of a student must also be a member of the thesis committee for that student.

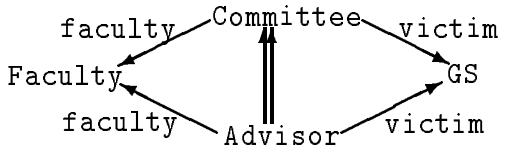

Figure 9: "Isa-A" Relation Between Relationships

Since the committee for a student consists of several faculty members, and each faculty member can be on several thesis committees, the set of keys for `Committee` is {{`faculty, victim`}}. However, since each student has at most one advisor, but that each faculty member

can be the advisor of several students, the set of keys for `Advisor` is {{`victim`}}. This is equivalent to having cardinality constraints asserting that the relation `Advisor` is "one-to-many" while `Committee` is "many-to-many". Note that {{`victim`}, {`faculty, victim`}} $\supseteq$ {{`faculty, victim`}}, thus the merged schema satisfies our constraint.

Our task now becomes to derive keys in the merged schema subject to this constraint.

Suppose schema $\mathcal{G}$ is the proper schema merge of $\mathcal{G}_1$ and $\mathcal{G}_2$. Each class p in $\mathcal{G}$ appears at most once in each of $\mathcal{G}_1$ and $\mathcal{G}_2$, with key assignments $\mathcal{SK}_1(p)$, $\mathcal{SK}_2(p)$ respectively (when defined). We define $\mathcal{SK}$ to be a *satisfactory assignment* of keys to classes if

1. $\mathcal{SK}_1(p) \subseteq \mathcal{SK}(p)$, if $p \in \mathcal{C}_1$; and
2. $\mathcal{SK}_2(p) \subseteq \mathcal{SK}(p)$, if $p \in C_2$; and
3. $\mathcal{SK}$ satisfies the condition that $\mathcal{SK}(p) \supseteq \mathcal{SK}(q)$ whenever $p \Longrightarrow q$.

It is readily checked that if $\mathcal{SK}$ and $\mathcal{SK}\prime$ are satisfactory assignments, then so is $\mathcal{SK} \cap \mathcal{SK}\prime$, defined by $(\mathcal{SK} \cap \mathcal{SK}\prime)(p) = \mathcal{SK}(p) \cap \mathcal{SK}\prime(p)$. Thus there is a unique minimal satisfactory assignment of keys to classes.

We can see that key constraints are sufficient to capture the kinds of cardinality constraint most commonly found in the ER literature, namely the restriction of attributes or relations to being "many-to-many", "many-to-one" and so on, at least in the case of binary relationships where their meaning is clear. However they are not capable of representing the participation constraints, representing total versus partial participation of an entity set in a relationship, found in some models (see [3]): for example, we cannot use key in our schemas to specify that each graduate student *must* have a faculty advisor, but that not every faculty member must necessarily advise some student. On the other hand, cardinality constraints cannot capture all key assertions: For example, consider the relationship `Transaction` in Figure 10. The statement that `Transaction` has two keys, one being {`loc, at`}, the other being {`card, at`}, has no correspondence in terms of labeling edges.

Keys can also be used to determine when an object in the extent of a class in an instance of one schema corresponds to an object in the extent of the same class in an instance of another schema. For example, if `Person` is a class in two schemas, $\mathcal{G}_1$ and $\mathcal{G}_2$, which are being merged,

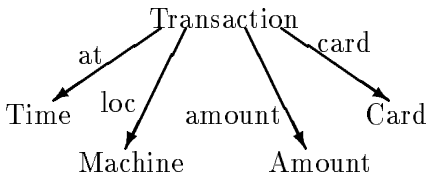

Figure 10: A Class with Multiple Keys

and both schemas agree that {SS#} is a key for Person, then an object in the extent of Person in an instance of $\mathcal{G}_1$ corresponds to an object in the extent of Person in an instance of $\mathcal{G}_2$ if they have the same social security number. However, suppose that $\mathcal{G}_1$ claims that {SS#} is a key for Person, and $\mathcal{G}_2$ has an SS#-arrow for Person but does *not* claim that it is a key. Since {SS#} is a key for Person in the merged schema, an additional constraint has been placed on the extents of $\mathcal{G}_2$: two objects in the extent of Person are the same if they have the same social security number, no matter whether both are from an instance of $\mathcal{G}_1$, both are from an instance of $\mathcal{G}_2$, or one is from an instance of $\mathcal{G}_1$ while the other is from an instance of $\mathcal{G}_2$. Furthermore, if $\mathcal{G}_1$ claims that {SS#} is a key for Person but $\mathcal{G}_2$ does *not have* an SS#-arrow for Person, then there is not way to tell when an object from the extent of Person in an instance of $\mathcal{G}_1$ corresponds to an object from the extent of Person in an instance of $\mathcal{G}_2$.

6 Lower Merges

In Section 4 we defined the merge of a collection of schemas as their *least upper bound* under an information ordering. A consequence of this is that, if we merge a number of schemas, then any instance of the merged schema can be considered to be an instance of any of the schemas being merged. In some cases, however, it is desirable to find the *greatest lower bound* of a collection of schemas and use that as the merge. In this case any instances of the schemas being merged would also be instances of the merged schema, and, further, we would expect to be able to coalesce or take the union of a number of instances of the collection of schemas and use that as an instance of the merged schema. This kind of merge is likely to arise in,

for example, the formulation of federated database systems.

We will refer to the merges defined in section 4 as **upper merges**, and, in this section, we will discuss the formulation of **lower merges**, representing the greatest lower bound of a collection of schemas. It could legitamately be argued that lower merges are of primary importance and should have been introduced first. However we introduced upper merges as our primary concept of a merge because they are inherently simpler and more natural to formulate. There are a number of complications involved in giving a formal definition of lower merges. For a detailed treatment of the problems involved see [5].

As it stands, taking the lower bound of a collection of schemas using our information ordering is clearly unsatisfactory: any information on which two schemas disagree on is lost. For example if one schema has the class *Dog* with arrows *name* and *age*, and another has *Dog* with arrows *name* and *breed*, then in the lower bound of the two schemas the class *Dog* will only have the arrow *name*. What we want, however, is some way of saying that instances of the class *Dog may* have *age*-arrows and *may* have *breed*-arrows, but are not necessarily required to do so. Worse still, if one schema has the class *Guide-Dog* and another does not, then the lower bound of the two schemas will not. The second problem can be dealt with easily by adding all classes involved in other schemas to each schema in a collection before proceeding with the construction of the lower merge. The first problem, however, is more difficult and requires us to extend our definition of (weak) schemas.

We define the semi-lattice of **participation constraints**, ordered by $\leq$, to be as shown in figure 11. We will extend the definition of (weak) schemas by associating a participation

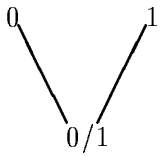

Figure 11: The semi-lattice of participation constraints

constraint with each arrow of a schema via a mapping $\mathcal{K} : \mathcal{E} \to \{0, 0/1, 1\}$. The idea is that,

if a class p has an a-arrow of class q, then if the arrow has participation constraint 1 then every instance of class p *must* have a an a-arrow to an instance of class q; if the arrow has participation constraint 0/1 then an instance of p *may* have an a-arrow of class q; and if the arrow has constraint 0 then an instance of p *may not* have an a-arrow of class q. We adopt the convention of not drawing arrows with the participation constraint 0 in our diagram, and, further, assume that a schema which does not have some arrow $p \xrightarrow{a} q$ is equivalent to the same schema but with the arrow $p \xrightarrow{a} q$ with participation constraint 0.

Now, if one schema has an arrow which is not included in another, then we can assume that the second schema also has the arrow, but with participation constraint 0, and we can take the greatest lower bound of the participation constraints (under the ordering $\leq$) to be the participation constraint of the arrow in the merged schema.

Hence, with the addition of participation constraints, we can form the *weak lower merge* of a collection of schemas in a similar manner to that used to construct the *weak upper merges* in section 4. We can also build a proper schema from a weak lower merge using an algorithm similar to that in section 4, except that that the implicit classes are introduced above, rather than bellow, the sets of proper schemas that they represent.

It is worth noting that, while upper and lower merges represent two extreme views of what the merge of a collection of schemas should be, there may well be valid and useful concepts of merges lying inbetween the two. However the authors believe that, in order for a concept of a merge to be valid and well defined, it should have a definition in terms of an information ordering similar to the ones given here.

7 Conclusions

Using a simple but general formalism, we have characterized the *weak schema merge* of a collection of schemas as their least upper bound. The *merge* of these schemas is then defined by translating the weak schema merge to a *proper* schema. The translation introduces new "implicit" classes as required, and identifies their origin in their name. Although not discussed in detail in this paper, the "real-world" validity of an implicit class can be efficiently checked by consulting a consistency relationship between the classes from which the implicit class was

formed.

Despite the simplicity of our mathematical construction, we believe that using an information ordering is the right way of describing the merge of schemas: it has a well-defined result, and the merge operation is associative and commutative. Thus user assertions about the relationships between schemas can be thought of as real assertions rather than "guiding heuristics" since the merge is independent of the order in which the assertions are stated. The approach in this paper focused on the upper merge of schemas, which seems to be the most natural concept of a merge. Other kinds of merge can be defined, including the lower merge, by varying the information ordering used.

The approach presented in this paper can be generalized to describe the merge in a number of other data models by representing schemas in other data models as "restricted" instances of schemas in our general model (*i.e.* stratifying classes in terms of their meaning in other models), and finding their proper schema merge. Our merge can be shown to "preserve strata", guaranteeing that the result will an instance of the original model; a proof of this with full details can be found in [5].

To use this approach as a practical schema merging tool, several issues should be addressed. Firstly, more attention should be paid to how cardinality constraints should be encorporated. While our preliminary approach has been to use a notion of keys, other ideas include allowing arrows to be "multivalued functions" as in [2]; [5] shows how this idea can be extended to our model. Secondly, some form of assistance should be given for "restructuring" schemas to obtain a better merge. Not only can "naming" conflicts occur (such as homonyms and synonyms), but "structural" conflicts can occur. For example, an attribute in one schema may look like an entity in another schema, or a many-one relationship may be a single arrow in one schema but introduce a relationship node in another schema. In these cases, the merge will not "resolve" the differences but present both interpretations. To force an integration, we need some kind of "normal form". Thirdly, we need to evaluate how many implicit classes can be introduced in the merge. Although in the examples we have looked at this number has been small, it may be possible to construct pathological examples in which the number of implicit classes is very large; however, we do not think these are likely to occur in practice. Fourthly, we must discuss how to merge instances; for a discussion of the problems involved, see [16].

We have found that the simplicity of the method and presence of strong theoretical underpinnings have made extensions of the technique very easy to develop. In addition, we have been able to rapidly prototype the method, together with a graphical interface for creating and displaying schema graphs.

References

[1] A. Motro, "Superviews: Virtual Integration of Multiple Databases," *IEEE Transactions on Software Engineering*, vol. SE-13, pp. 785–798, July 1987.

[2] J. Smith, P. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong, "Multibase–Integrating Heterogeneous Distributed Database Systems," in *Proceedings of AFIPS*, pp. 487–499, 1981.

[3] S. Navathe, R. Elmasri, and J. Larson, "Integrating User Views in Database Designs," *IEEE Computer*, pp. 50–62, January 1986.

[4] C. Batini, M. Lenzerini, and S. Navathe, "A Comparative Analysis of Methodologies for Database Schema Integration," *ACM Computing Surveys*, vol. 18, pp. 323–364, December 1986.

[5] A. Kosky, "Modeling and Merging Database Schemas," Tech. Rep. MS-CIS-91-65, University of Pennsylvania, 1991.

[6] D. Shipman, "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Database Systems*, vol. 6, pp. 140–173, March 1981.

[7] R. Hull and R. King, "Semantic Database Modeling: Survey, Applications, and Research Issues," *ACM Computing Surveys*, vol. 19, pp. 201–260, September 1987.

[8] A. Ohori, "Semantics of Types for Database Objects," *Theoretical Computer Science*, vol. 76, pp. 53–91, 1990.

[9] P. Chen, "The Entity-Relationship Model: Towards a Unified View of Data," *TODS*, vol. 1, no. 1, pp. 9–36, 1976.

[10] J. Ullman, *Principles of Database and Knowledge-Base Systems*. Vol. 1, Computer Science Press, 1988.

[11] H. Korth and A. Silberschatz, *Database System Concepts*. McGraw Hill, second ed., 1991.

[12] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*. Benjamin/Cummings, 1989.

[13] D. Tsichritzis and F. Lochovsky, *Data Models*. Prentice-Hall, 1982.

[14] T. Teory, D. Yang, and J. Fry, "A Logical Design Methodology for Relational Databases Using the Entity-Relationship Model," *ACM Computing Surveys*, vol. 18, pp. 197–222, June 1986.

[15] M. Lenzerini and G. Santucci, "Cardinality Constraints in the Entity Relationship Model," in *The Entity-Relationship Approach to Software Engineering*, pp. 529–549, North-Holland, 1983.

[16] S. Widjojo, R. Hull, and D. Wile, "Distributed Information Sharing Using WorldBase," in *A Newsletter of the Computer Society of IEEE*, pp. 17–26, August 1989.