A Basis for Interactive Schema Merging *

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

Abstract

We present a technique for merging the schemas of heterogeneous databases that generalizes to several different data models, and show how it can be used in an interactive program that merges Entity-Relationship diagrams. Given a collection of schemas to be merged, the user asserts the correspondence between entities and relationships in the various schemas by defining "isa" relations between them. These assertions are then considered to be elementary schemas, and are combined with the elementary schemas in the merge. Since the method defines the merge to be the join in an information ordering on schemas, it is a commutative and associative operation, which means that the merge is defined independent of the order in which schemas are presented. We briefly describe a prototype interactive schema merging tool that has been built on these principles.

Keywords: *schemas, merging, semantic data models, entity-relationship data models, inheritance*

1 Introduction

Schema merging is the problem of taking several database schemas and combining them into a single user view. This problem may arise either in the initial design phase of a large database application, or as an afterthought to provide integrated access to a collection of previously existing databases. In the former application, several teams of designers may be given the specification of different subsets of the database application, and asked to construct sub-schemas in a common model that meet the specifications. The sub-schemas are then to be combined into a single global schema. In doing so, competing design insights and interpretations of the teams must be mediated to form a coherent schema (see [1]). For example, there may be conflicts of *name*, either homonyms or synonyms; conflicts of *scale*, where the underlying domains of a commonly named attribute are not the same, although there may be a simple translation function between the domains; *structural* conflicts, an example of which is when an entity in one schema is considered to be an attribute of another entity in another schema; and different *levels of abstraction*, in which one schema presents more detailed information than another. The problem of providing integrated access to a collection of previously existing databases must resolve the same types of conflict, and additionally perform some sort of translation from potentially different underlying database models to a common intermediary model.

Various approaches to schema merging have been proposed; see [2] for a survey. These vary from sets of tools for manipulating two schemas into some form of consistency ([3, 4]), to algorithms which take two schemas together with some constraints and produce a merged schema. In practice a method that lies somewhere between these two extremes is usually desirable: a certain amount of user intervention is inevitable, though, once any conflicts have been resolved and the correspondences between the various elements of some database schemas have been identified, it is helpful to have an automated merging algorithm, especially for large schemas.

In this paper, we address the problem of what meaning or semantics the merging process should have, and describe a merging tool based on this semantics. We develop a simple and general characterization of database schemas that allows us to define the merged schema in terms of the informational content of the schemas being merged. Using an information ordering on schemas, we define the merge of a collection of schemas to be their *least upper bound* (or join): The merge takes the union of all information

*This research was supported in part by ARO DAAL03-89-C-0031PRIME and NSF IRI 8610617, as well as by a grant from the UK SERC at Imperial College, London.

stored in the database schemas, and, wherever possible, forms a schema presenting this but no additional information. User interaction is incorporated into the process by allowing the user to specify an integration schema which captures the relationship between structures in the various schemas; the integration schema is then combined with the elementary schemas in the merge. Since the method defines the merge to be the *join* of the elementary and integration schemas, it is a commutative and associative operation. That is, the merge is defined *independent of the order in which schemas are presented*. Therefore, whether a collection of N schemas is integrated in a one-shot or iterative n-ary merge, or as a succession of ladder or balanced binary merges [5] is irrelevant since the outcome will be the same.

For an example of the importance of these mathematical properties we cite Navathe *et al.*[6], who working in the context of the "Entity-Category-Relationship" model, discuss an interactive merging process which is "phased". They also suggest that each phase should be carefully controlled by the user, who supplies assertions about the components of the schemas to be merged. In this framework there are two reasons for a merging operation with clean mathematical properties: first, the semantics of a database schema can be extremely complicated, and there is every chance that a user assertion will prove incorrect. To "undo" the effect of an assertion it is highly desirable to have a process that depends only on the *set* of assertions, and not on the order in which they were presented. Associativity and commutativity (and idempotence) guarantee this. Lack of associativity leads to situations in which the only sensible way to undo a previous assertion is to "back out" of all the changes that were made since that assertion.

The second reason is one of simplicity; in the method we propose here, assertions that connect schemas may themselves be regarded as "elementary" schemas. The merging process now has a much simpler description: take the input schemas, together with the user assertions and merge everything. To elaborate on these points let us briefly look at some examples drawn from the Entity-Relationship model.

The process described in [6] starts from user-supplied assertions about the object classes (the entities and their sub-classes) and, once this has been completed, proceeds to integration of the relationships. Finding the correct assertions is a delicate matter, and an incorrect choice can have a dramatic result on the merged schema. Figure 1 (a modification of an example in [6]) shows an example of a merge. Figure 2 shows a similar example – the correspondences between the names of entities and relationships are identical – but the result is different. To understand that both results are possible, consider a simple "object-oriented" semantics which associates with each entity or relationship X, a set $[\![X]\!]$ of "real-world" instances of X, with each edge from relationship R to entity-set E, a function in $[\![R]\!] \to [\![E]\!]$, and with each specialization (between entities or relationships) $X_1 \Rightarrow X_2$, an inclusion $[\![X_1]\!] \subseteq [\![X_2]\!]$. In figure 1, while we expect that $[\![\text{Course}_1]\!] = [\![\text{Course}_2]\!]$

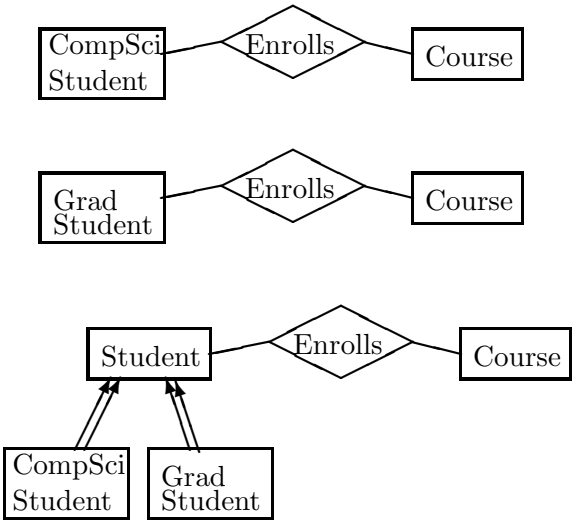

Figure 1: An example merge

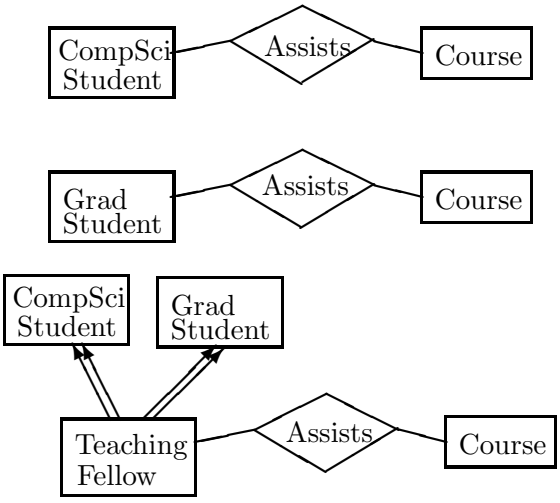

Figure 2: Another similar example

it is not the case that $[\![\text{Enroll}_1]\!] = [\![\text{Enroll}_2]\!]$. By contrast, in figure 2, it is reasonable to assert that $[\![\text{Assists}_1]\!] = [\![\text{Assists}_2]\!]$. Another way of understanding this difference is to consider how the component databases could have been produced as views of a hypothetical database. The real-world sets associated with the entities and relationships are preserved in the views, so that in figure 2 $[\![\text{Assists}_1]\!] = [\![\text{Assists}]\!]$ and $[\![\text{Graduate Student}_1]\!] = [\![\text{Graduate Student}]\!]$. In figure 1, the best we can say is that the real-world set $[\![\text{Enroll}_1]\!]$ is some subset of $[\![\text{Enroll}]\!]$.

Note that assertions about the equivalence of real-world sets can be represented by schema fragments. Containment between real-world sets are represented by specialization arrows, and identification of entities or relationships could be represented by double specializations, so that in example two, the fact that $[\![\text{Assists}_1]\!] = [\![\text{Assists}_2]\!]$ could be represented by the two elementary schemas $\text{Assists}_1 \Rightarrow \text{Assists}_2$ and $\text{Assists}_2 \Rightarrow \text{Assists}_1$.

These examples demonstrate the problems of identifying entities or relationships because of their names. If we make such an identification then figure 2 represents the correct merge, and this is what our method will do. However a case can be made that similar names indicate only that the real-world sets overlap, in which case some alternative strategy must be adopted. A "lower" merge, which yields the results of figure 1, is discussed in [7, 8]. We do not address the question of resolving naming conflicts, since the problem seems to be inherently *ad hoc* in nature, nor do we address the problem of structural conflicts, although the simplicity of our model may eliminate many of these conflicts. Furthermore, since the merge takes the *union* of all information stored in the database schemas, the problem of different levels of abstraction is also addressed to some extent.

The remainder of this paper is organized as follows: Sections 2 and 3 describe the model and merging technique. In section 4 we describe a prototype schema merging tool implementing the method. Section 5 describes how our model can be thought of as "subsuming" other data-models, such as the relational, functional and ER models, and shows in what sense the merging process respects the original model.

2 The Model

In order to define a merging operation we need first to formulate a data model. The model we shall work with is explained in detail in [7, 8]; we briefly motivate it and review it here. Data models are generally graphical structures consisting of nodes and edges. For example, the ER model, in its basic form, has three node shapes (for Attribute domains, Entities and Relationships). We shall refer to these nodes – whatever their shape – as *classes*. The edges in the ER model are subject to certain restrictions and are often, but not always, labeled. We shall allow our edges, which we call *arrows* to connect arbitrary classes, and they will always be labeled. We shall also include *specialization* relations between classes in our model.

Formally, we assume universal sets $\mathcal{N}$ of *classes* and $\mathcal{L}$ of *arrow labels*. A database schema then consists of a triple $(\mathcal{C}, \mathcal{E}, \mathcal{S})$ where $\mathcal{C} \subseteq \mathcal{N}$ are the classes of the schema, $\mathcal{E} \subseteq \mathcal{C} \times \mathcal{L} \times \mathcal{C}$ is the set of labeled arrow edges, and $\mathcal{S}$is a partial order (that is a transitive, reflexive and antisymmetric relation) on $\mathcal{C}$. If we have an edge $(p, a, q) \in \mathcal{E}$ then we write $p \xrightarrow{a} q$ and say that "p has an a-arrow of class q". If a pair of classes (p, q) is in the relation $\mathcal{S}$ then we write $p \Longrightarrow q$, and say that "p is a specialization of q" (the idea here is that p is a more specific class than q, so that every instance of p can also be considered to be an instance of q). In addition we require that $\mathcal{E}$ and $\mathcal{S}$satisfy the following axioms:

1. If $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ then $q = r$.
2. If $q \xrightarrow{a} s$ and $p \Longrightarrow q$ then there is an $r \in \mathcal{C}$ such that $r \Longrightarrow s$ and $p \xrightarrow{a} r$.

The first constraint here says that any particular class can have at most one a-arrow for any arrow label a, so that, for example, a class `Person` could not have a `name`-arrow of class `str` and another `name`-arrow of class `int`. The second restriction says that arrows are, in some sense, preserved by specialization, so that, if p is a specialization of q and q has an a-arrow of class s, then p also has an a-arrow with class at least as specific as s. These axioms are equivalent to those given for functional schemas in [4] and also in [3] (though the latter used unlabeled arrows).

Let us consider the ER diagram of a University Business Office database shown in figure 3. The diagram indicates that there are two ways of classifying University Employees (`UE`): as research employees (`Res_UE`) or as academic employees (`Ac_UE`). Teaching Fellows (`TF`) are classified exclusively as academic employees (indicated by `TF`$\Longrightarrow$`Ac_UE`), while Research Fellows `RF` are classified exclusively as research employees (indicated by `RF`$\Longrightarrow$`Res_UE`). Faculty members, however, are classified as both research and academic employees (indicated by `Faculty`$\Longrightarrow$`Res_UE` and `Faculty`$\Longrightarrow$`Ac_UE`). While there is a single budget item from which academic employees are paid, re-

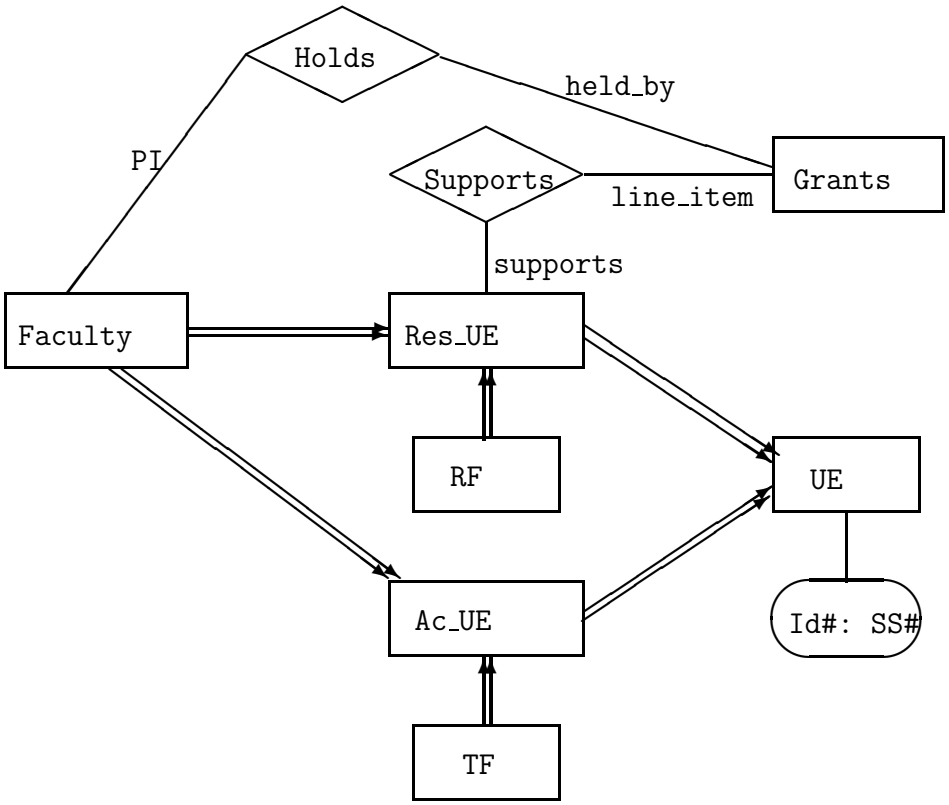

Figure 3: Business Office ER Diagram

search funds are provided by grants held by faculty members (indicated by the relationships `Holds` and `Supports`). Much detail has been omitted; in particular, we have omitted almost all attributes of entities and relationships to avoid cluttering the diagram.

The corresponding database schema for the Business Office, in our model is shown in figure 4, 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).

We can use our model to represent the schemas of a number of other data models. We have already seen how we can interpret an entity-relationship schema as a schema in our model: basicly the ER-model can be considered to be a restriction of our model where the set of classes, $\mathcal{C}$, is stratified into three sets, $\mathcal{C}_R$, $\mathcal{C}_E$ and $\mathcal{C}_D$, corresponding to relationships, entity-sets and attribute domains respectively, and arrows are restricted so that they can only go from classes in one set to classes in the next. Similarly the relational data model can be thought of as stratifying the classes of our model into two sets: relations and domains. We will discuss such restrictions in more detail in section 5, and will show that the merging process described in the next section respects these restrictions. By a less constrained process we can describe instances of the functional model [9, 4, 3]. The graphs are also general enough to represent databases with higher order relationships (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, our model does in some sense subsume a number of established data-models, and by investigating the problem of schema merging for this model we can understand how to merge schemas in these other models. However, as it stands, the model is not sophisticated enough to represent the variants and set-valued attributes occurring in certain models such as those proposed in [10] and [11], and would

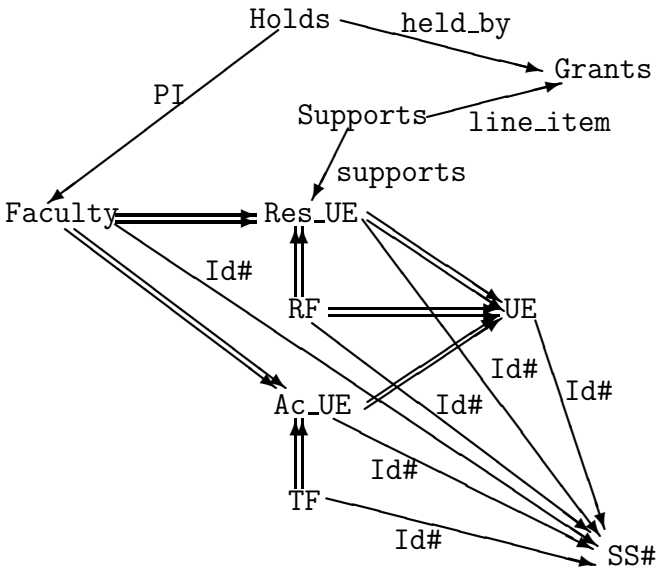

Figure 4: Business Office Schema

require further extensions to do so.

3 The Method

When merging a collection of schemas our first task must be to decide on the correspondences between the classes and arrows of the various schemas. These correspondences are inherently dependent on the real-world interpretation of the schemas, and so must be determined by the designer of the system before the merge can proceed. Although there is scope for the use of various sophisticated tools to help the designer identify such correspondences, we take the simplistic approach that two classes occurring in different schemas correspond to the same real world class of objects if and only if they have the same class name, regardless of whether or not they have the same arrows. For example, if one schema has a class `GS` with arrow edges `Id#`, `Name` and `Status`, and another schema has a class `GS` with arrow edges `Name`, `Address` and `Telephone`, then the merging process will collapse them into a single class with name `GS` and arrow edges `Id#`, `Name`, `Status`, `Address`, and `Telephone`. In addition we allow the designer of a system to assert that a class, p, in one schema is a specialization or subclass of another class, q, in a second schema, that is $p \Longrightarrow q$ in the merged schema, by adding an "atomic" schema $p \Longrightarrow q$ to the collection of schemas being merged.

The philosophy we adopt is that the merge of a collection of schemas is a schema which presents all the information of the schemas being merged, but no additional information. Hence the merge is the "least upper bound" of the database schemas under some sort of information ordering. Recall that, in addition to defining a view of a database, a database schema expresses certain requirements on the structure of the information stored in the database. When we say that one database schema presents more information than another, we mean that any instance of the first schema could be considered to be an instance of the second one. The first schema must, therefore, dictate that any database instances must contain at least all the information necessary in order to be an instance of the second schema. It is clear that, were we to construct an ordering on schemas which represented the fact that one schema contained more information than another, it should be a partial ordering on schemas, and the binary merge of two schemas would be their least upper bound, or join, under this ordering. This ties in well with our intuition that a binary merging operator on schemas should be associative, commutative and idempotent.

It turns out that the first constraint on schemas in section 2, which requires that each a-arrow out of p has a unique class, significantly complicates finding such an ordering (see [7] for more details). We therefore ini-

tially weaken our definition of database schemas, and define an information ordering on these *weak schemas* in which least upper bounds exist and are associative. The *weak schema merge* is then defined to be the least upper bound in this ordering. Since the weak schema merge is not guaranteed to meet all conditions on *proper schemas* (as we will henceforth refer to the schemas defined in section 2), we must then convert the weak schema merge to a proper schema by introducing new classes for a-arrows out of each node that violates the first constraint on proper schemas.

We start by discussing the weak schema merge, and then describe how to convert a weak schema to a proper schema.

3.1 Weak Schemas

A *weak schema* is a schema in which we no longer require that, for any arrow label a, a class p can have at most one a-arrow (condition 1 of proper schemas), but instead require the weaker condition that, for any class p and arrow label a, and any two distinct classes q and r, if $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$, then q and r may not be specializations of one another. 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 on $\mathcal{C}$, and $\mathcal{E}$ is a subset of $\mathcal{C} \times \mathcal{L} \times \mathcal{C}$ satisfying

W1. If $p \xrightarrow{a} q$ and $p \xrightarrow{a} r$ and $q \Longrightarrow r$ then $q = r$.

W2. If $p \Longrightarrow q$ and $q \xrightarrow{a} r$ then there is an $s \in \mathcal{C}$ such that $s \Longrightarrow r$ and $p \xrightarrow{a} s$.

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

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. If $p \xrightarrow{a}_1 q$ then there is an $r \in \mathcal{C}_2$ such that $p \xrightarrow{a}_2 r$ and $r \Longrightarrow_2 q$
3. $\mathcal{S}_1 \subseteq \mathcal{S}_2$

That is, every class in $\mathcal{G}_1$ appears in $\mathcal{G}_2$, for every a-arrow in $\mathcal{G}_1$ there is a corresponding arrow, with at least as specific a class, 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 order on weak schemas; it also has a property on least upper bounds described in the following proposition.

Proposition 3.1 *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 G_2 as above, define $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S})$ by

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

(where $(\mathcal{S}_1 \cup \mathcal{S}_2)^*$ denotes the transitive closure of $(\mathcal{S}_1 \cup \mathcal{S}_2)$, and $\mathcal{E}$ subtracts edges from $\mathcal{E}_1 \cup \mathcal{E}_2$ necessary to make condition W1 hold, and adds those necessary for condition W2 to hold). To complete the proof we must show that, if $\mathcal{G}$ is a weak schema, then $\mathcal{G}_1 \sqsubseteq \mathcal{G}$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}$, and that, if there is a weak schema $\mathcal{G}'$ such that $\mathcal{G}_1 \sqsubseteq \mathcal{G}'$ and $\mathcal{G}_2 \sqsubseteq \mathcal{G}'$, then $\mathcal{G}$ is indeed a weak schema and $\mathcal{G} \sqsubseteq \mathcal{G}'$. Details can be found in [7]. ∎

We can characterize those collections of schemas that are bounded above, and therefore have a least upper bound, by means of a simple restriction on their specialization relations. We say a finite collection of weak schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, satisfying this restriction is **compatible**. 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 the binary least upper bound operator defined in the proof of proposition 3.1 is associative and commutative, we can repeatedly apply it to pairs of schemas in a collection in order to get the merge of the entire collection.

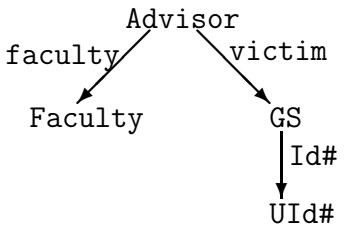

Figure 5: Department Office Schema

For example, suppose that we gain access to the Departmental Office schema shown in figure 5, which represents the advising relationship between Faculty and Graduate Students (`GS`), who are identified by their university id number (`UId#`). Since this schema

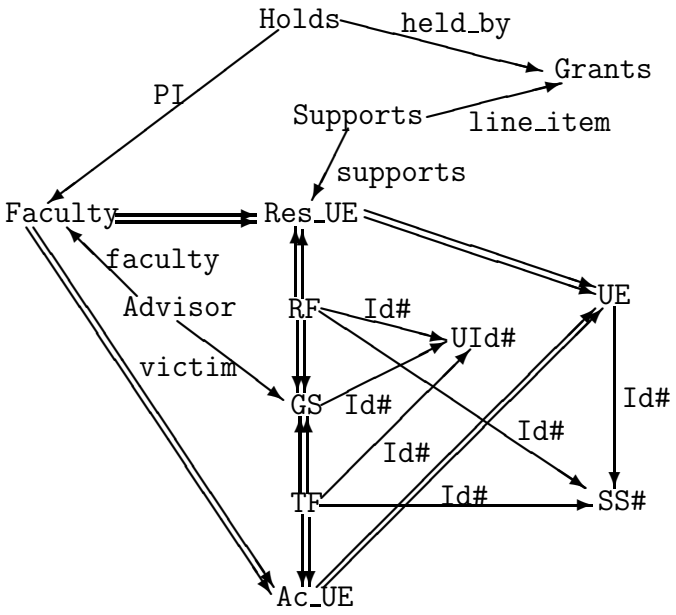

Figure 7: Weak Schema Merge of Departmental Office, Business Office and Integration Schemas

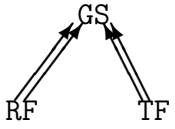

Figure 6: Integration Schema

contains information in common with the Business Office schema of figure 4, we decide to merge the two schemas. We know that there are no naming conflicts between the two schemas, although at first glance it might seem that the Id# label might be a candidate for renaming. However, we happen to know that a graduate student's university id number is their social security number unless they do not have one, in which case a unique internal number is assigned. We also know that every teaching fellow is a graduate student, and that every research fellow is a graduate student; these constraints are specified in the integration schema shown in figure 6. Taking the weak schema merge of the Departmental and Business Office schemas, together with the integration schema, we get the schema shown in figure 7.

Note that to avoid cluttering the figure we have omitted the `Id#`-arrows from `Res_UE`, `Ac_UE` and `Faculty` to `SS#`. Also note that the merged schema is *not* proper since there is no canonical class for the `Id#`-arrow of `RF` or `TF`.

3.2 Building proper schemas from weak schemas

Having found the weak schema merge, $\mathcal{G}$, of a collection of schemas, we must then find some way of introducing implicit classes into $\mathcal{G}$ in order to make it into a proper schema. What we need to find is 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. The algorithm to do this basically works by looking for violations of condition 1 of proper schemas, and correcting them by introducing new *implicit* classes. That is, if an a-arrow for some class p in the merged schema has more than one class, an *implicit* class labeled with all the names of the classes of the a-arrow is created. $\mathcal{S}$ is then augmented to indicate that the implicit class is a specialization of all classes of a-arrows out of p, and the a-arrows from p in $\mathcal{E}$ are replaced by a single a-arrow from p to the new implicit class. An implicit class may also inherit a-arrows with more than one class from the classes with which it is labeled, in which

case the process must be repeated. Since the derivation of the implicit class is contained in its name, this information can be used in subsequent merges to ensure associativity.

Details of the algorithm can be found in [7]; although it looks slightly complicated it is well specified and easily automated.

For example, to convert the weak schema in figure 7 to a proper schema, we would introduce the implicit class $\overline{\{UId\#, SS\#\}}$, and augment $\mathcal{S}$ with $\overline{\{UId\#, SS\#\}} \Longrightarrow UId\#$, $\overline{\{UId\#, SS\#\}} \Longrightarrow SS\#$, and replacing the $Id\#$-arrows from TF and RF in $\mathcal{E}$ by $TF \xrightarrow{Id\#} \overline{\{UId\#, SS\#\}}$ and $RF \xrightarrow{Id\#} \overline{\{UId\#, SS\#\}}$.

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 $\mathcal{C}$, and require that, for every pair of classes p and q in the label of an implicit class, (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.

4 An Implementation

To demonstrate the feasibility of our approach, we have developed a prototype. The program, called Xmerg, uses an X Window System interface for creating, displaying, and manipulating schema graphs. When Xmerg is invoked, one sees three major areas on the screen. On the left is a control panel with various buttons that are used to create and do operations on schemas. On the bottom is a long thin window in which messages to the user are printed out. Occupying the main area of the program's display is a large rectangle (the canvas) in which the schema graphs are drawn.

For simplicity in drawing the graphs, the specialization edges are drawn with dotted lines (rather than with double lines), and the arrowheads are replaced by small squares. In order to avoid clutter on the screen specialization edges implied by the transitivity of the specialization relation, and arrows implied by the second constraint on proper and weak schemas, are not drawn. In figure 8 Xmerg is shown with the Departmental Office, Business Office, and Integration schemas drawn in this manner.

4.1 Using Xmerg

The user creates schemas (here a schema is a connected component of the graph) using the *class, specialization*, and *attribute* (used for drawing arrows) buttons. For the *class* and *attribute* operations, a pop-up box appears in which one types the name of the class or label of the arrow.

The *move* button allows the user to move a schema to a new place.

The *merge* button begins the process of merging. The user selects all the schemas that will be merged (by clicking on one class from each) and then clicks on the *merge* button again to execute the operation. The merged weak schema is computed and converted to a proper schema by adding implicit classes and their associated edges, and then a version of this proper schema, omitting the specialization edges and arrows implied by transitivity of the specialization relation and the second constraint on schemas respectively, is displayed on the screen and is available for further manipulation.

The *check* operation checks to see if all the schemas on the canvas (with omitted arrows reinstated) are proper schemas. Two things could go wrong: the specialization relationship could fail to be antisymmetric (that is, following specialization edges there is a loop in the graph) or some class might have multiple a-arrows for some label a.

The *save* operation allows the user to save all the information on the canvas into a file. The *read* retrieves all information from a file created with the *save* command, replacing anything that was already on the canvas.

The *quit* operation, when selected while another operation is in session (that is, while the user is expected to click the mouse on the canvas to complete or continue an operation) will cancel that operation. If there is no operation in session, the program will be killed.

In figure 9, the merged version of the University Schemas is shown as computed and displayed by Xmerg.

Note that the merged schema looks more or less like what you would get if you moved around the graphs to place the same-named classes on top of one another. This is in fact just what has happened. Two schemas

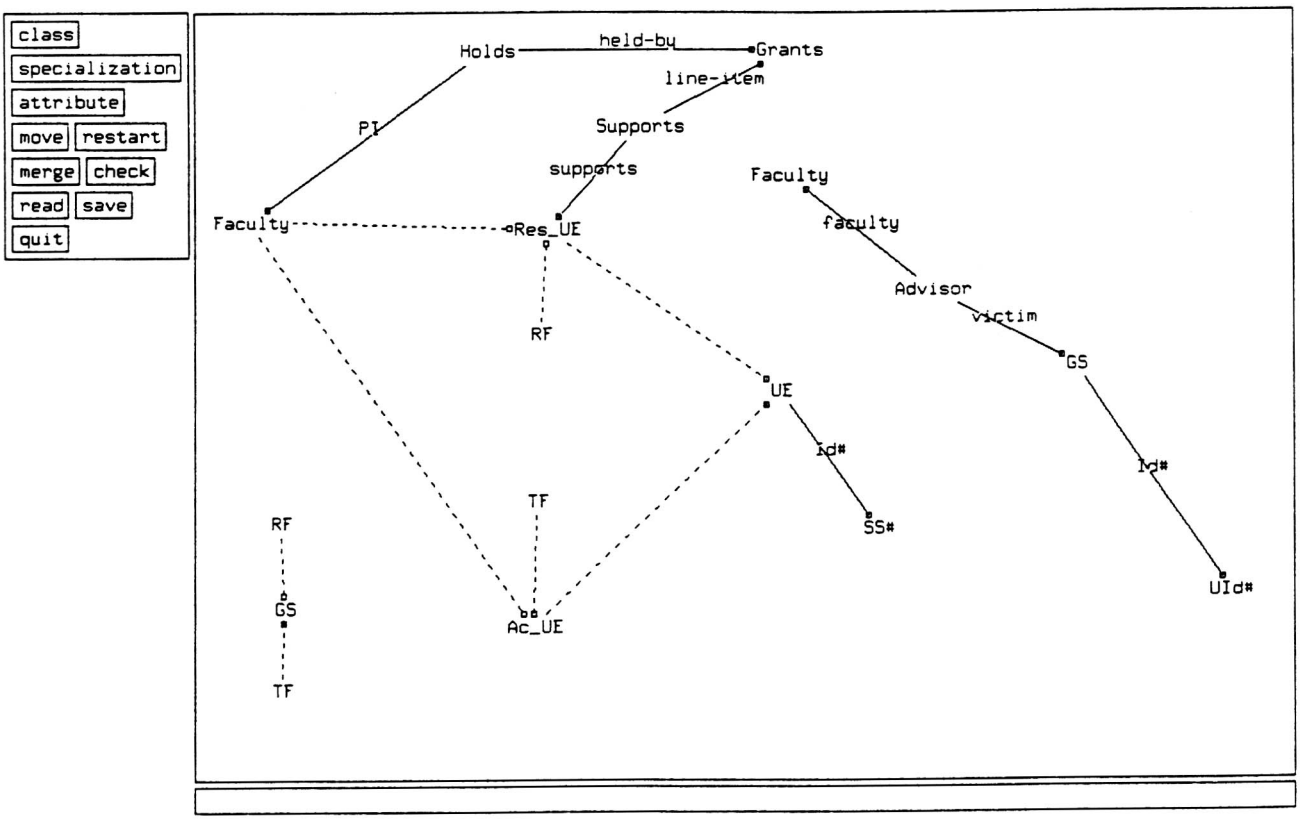

Figure 8: University Schemas as they would appear in Xmerg

are merged by finding a class they have in common and moving the second schema so that the two classes coincide. Then for each class in the second schema that has a class of the same name in the first schema, edges going to and from the class in the second schema are reassigned to go to and from the corresponding class in the first schema.

4.2 Fixes and Additions

The program as it currently stands is intended merely for experimental and demonstration purposes, rather than as a serious tool for schema merging. In order to make such a tool we would extend the user interface with a number of features in order to make it easier to draw and modify schemas. In addition some mechanism for recording which classes of a schema were introduced as implicit classes and how they arose would be required, which would allow further merges to be done on the merged schema.

Most of the schemas we have been working with are "toy", as found throughout this paper. To make the tool useful for larger schemas, we will have to use a sophisticated graph layout tool or adapt Xmerg to do much fancier graphical manipulations. For example, it would be useful to be able to handle more than one edge between two classes, to have a more sophisticated method of drawing the merged schema than approximately the way the component schemas are drawn, to allow users to zoom in or out on portions of the schema graph, and to scroll over a schema graph.

5 Meta-schemas

We demonstrated in Section 2 that ER schemas can be embedded in our model in a natural way. We then went on in Section 3 to find a way of determining the merge of a collection of schemas whenever it exists. In order for this conversion-and-merging process to be useful for ER schemas, we must show that the schema resulting from such a merge can also be considered to be an embedding of an ER schema. We could then reverse the conversion process in order to find an ER schema which is the merge of our original collection of ER schemas.

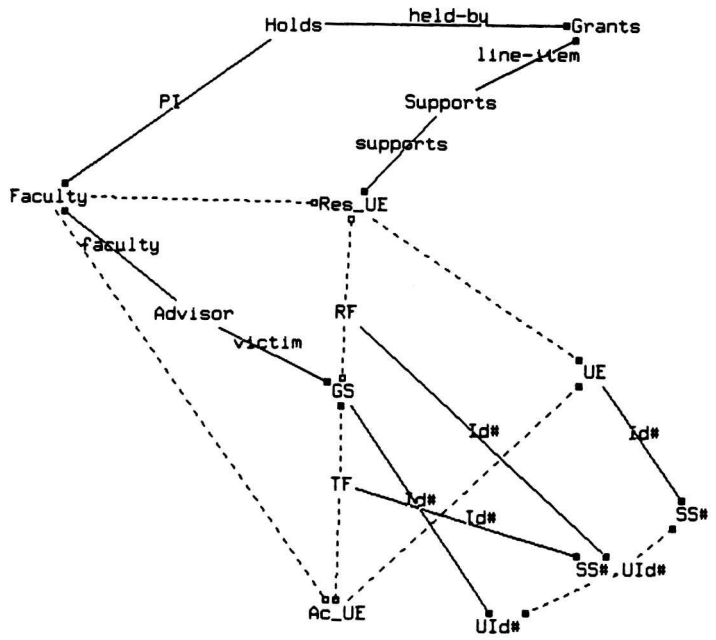

Figure 9: Merged University Schemas

Compared to our model, the most significant restriction in the ER model is that classes are stratified into three disjoint sets: entity sets, relationships and base types. Furthermore, specialization relationships are constrained so that they may only occur between classes in the same stratum, and the arrows of a class belonging to one stratum must have classes in the next stratum.

In this section we will look at the problem of imposing such constraints on schemas, and will show that these constraints are in some sense preserved by our merging process. While ER-like models always divide classes up into three distinct strata and only allow arrows to go from one stratum to the next, we will present a more general form of constraint, which we will call a *meta-schema*, which divides the classes into any number of distinct sets and then imposes restrictions on which arrows may go from classes in each set to classes in each other set.

A **meta-schema** over classes $\mathcal{N}$ and arrow labels $\mathcal{L}$ consists of a pairwise disjoint collection of sets of classes, $\{\mathcal{C}_i \mid i \in I\}$, such that $\mathcal{N} = \bigcup_{i \in I} \mathcal{C}_i$, and a collection of sets of labels $\{\mathcal{L}_{i,j} \mid i, j \in I\}$ such that, for each $i \in I$, the collection of sets of labels $\{\mathcal{L}_{i,j} \mid j \in I\}$ is pairwise disjoint.

A schema $\mathcal{G} = (\mathcal{C}, \mathcal{E}, \mathcal{S})$ is said to **satisfy** a meta-schema $(\{\mathcal{C}_i \mid i \in I\}, \{\mathcal{L}_{i,j} \mid i, j \in I\})$ iff

1. If $p \Longrightarrow q \in \mathcal{S}$ then $p \in \mathcal{C}_i$ and $q \in \mathcal{C}_i$ for some $i \in I$.
2. If $p \xrightarrow{a} q \in \mathcal{E}$ then $p \in \mathcal{C}_i$, $q \in \mathcal{C}_j$ and $a \in \mathcal{L}_{i,j}$ for some $i, j \in I$.

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

So the classes are divided into the sets $\{\mathcal{C}_i\}$, and two classes can only be related by a specialization relation if they belong to the same set, and, in addition, any class in a set $\mathcal{C}_i$ can only have an a-arrow with a class in the set $\mathcal{C}_j$ if a is in the set of labels $\mathcal{L}_{i,j}$.

For example, the schema shown in figure 4 satisfies the meta-schema given by taking $I = \{0, 1, 2\}$ and

$$\begin{aligned}
\mathcal{C}_0 &= \{\texttt{SS\#}\} \\
\mathcal{C}_1 &= \{\texttt{Faculty, Res_UE, Ac_UE, RF, TF}\} \\
\mathcal{C}_2 &= \{\texttt{Holds, Supports}\} \\
\mathcal{L}_{1,0} &= \{\texttt{Id\#}\} \\
\mathcal{L}_{2,1} &= \{\texttt{PI, held_by, supports, line_item}\}
\end{aligned}$$

and $\mathcal{L}_{i,j} = \emptyset$ for all other $i, j \in I$. In general any ER-structure will translate to a schema satisfying a meta-schema of the form found in figure 10.

Suppose we have meta-schema $\mathcal{M}$ and a consistent collection of proper schemas, $\mathcal{G}_1, \ldots, \mathcal{G}_n$, collectively satisfying $\mathcal{M}$. We would like to show that, if $\mathcal{G} = \bigsqcup_{i=1}^{n} \mathcal{G}_i$, then $\overline{\mathcal{G}}$ is a proper schema satisfying $\mathcal{M}$.

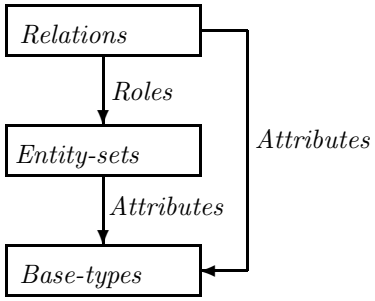

Figure 10: A meta-schema for ER-structures

Of course this is not quite true since $\overline{\mathcal{G}}$ may contain additional classes not in $\mathcal{C}$, however we do get the following theorem:

Theorem 5.1 *If* $\mathcal{M} = (\{\mathcal{C}_i \mid i \in I\}, \{\mathcal{L}_{i,j} \mid i,j \in I\})$ *is a meta-schema over* $\mathcal{N}$ *and* $\mathcal{L}$*, and* $\mathcal{G}_1, \ldots, \mathcal{G}_n$ *is a consistent collection of proper schemas collectively satisfying* $\mathcal{M}$ *such that* $\overline{\mathcal{G}}$ *is the merge of* $\mathcal{G}_1, \ldots, \mathcal{G}_n$*, then* $\overline{\mathcal{G}}$ *satisfies the meta-schema* $\overline{\mathcal{M}} = (\{\overline{\mathcal{C}}_i \mid i \in I\}, \{\mathcal{L}_{i,j} \mid i,j \in I\})$ *where*

$$\overline{\mathcal{C}}_i = \mathcal{C}_i \cup \{\overline{X} \mid X \subseteq \mathcal{C}_i\}$$

for $i \in I$*.*

Proof: Omitted. See [7] for details. ∎

For example, figures 4 and 5 collectively satisfy the following meta-schema $\mathcal{M}$:

$$\begin{aligned}
\mathcal{C}_0 &= \{\texttt{SS\#, UId}\} \\
\mathcal{C}_1 &= \{\texttt{Faculty, Res_UE, Ac_UE, RF,} \\
&\quad \texttt{TF, GS}\} \\
\mathcal{C}_2 &= \{\texttt{Holds, Supports, Advisor}\} \\
\mathcal{L}_{1,0} &= \{\texttt{Id\#}\} \\
\mathcal{L}_{2,1} &= \{\texttt{PI, held_by, supports,} \\
&\quad \texttt{line_item, faculty, victim}\}
\end{aligned}$$

and their proper schema merge $\overline{\mathcal{G}}$ satisfies $\mathcal{M}$ with C_0 augmented by the implicit class $\{SS\#, UId\#\}$.

It follows that if we use meta-schemas to constrain the schemas we are merging we can expect similar constraints to apply to the merge of those schemas.

6 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 into a *proper* schema. The translation introduces new "implicit" classes as required, and identifies their origin in their name. By keeping track of the implicit classes introduced into our schemas we can ensure that our merging process is associative and commutative. Consequently, the integration schema created by the user may treated as a collection of *assertions* about the relationships between the elementary schemas. 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; if an implicit class is created which violates the consistency relationship, either the elementary schemas are inconsistent or the integration schema is incorrect. In either case, the merge cannot take place and the error is reported to the user.

The problem of finding an upper bound on the number of implicit classes that are introduced in a merge remains open: there may be pathological examples where the number of implicit classes is exponential in the size of the schemas being merged, though we have not succeeded in creating such an example. In the examples used in existing work on the problem of schema merging, where implicit classes are introduced but not studied in detail or properly understood, the number of implicit classes is generally small. This fact, together with our own difficulty in finding realistic examples of merges where the number of implicit classes introduced is excessively large, leads us to believe that this will not be a problem in practice.

While the model introduced here is quite basic and lacks many of the features that adorn other data-models, it is easy to extend both the model and the merging process to incorporate such features. For example, in [8], we discuss how the model can be extended with *keys* and how these can be used to represent certain kinds of *cardinality constraint*; while, in [7] we show how set valued and null valued attributes can be represented.

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 was shown in section 5 to "preserve strata", guaranteeing that the result will an instance of the original model.

While implicit classes can be used to detect conflicts

of scale, structural conflicts are more difficult. 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 will present both interpretations. To force an integration, we need some kind of "normal form"; this is an area for future research. It is also worth noting that since the merge takes the *union* of all information stored in the database schemas, the problem of different levels of abstraction is also addressed to some extent. However, since null values may be introduced with this interpretation, a query language for such a system must be able to handle incomplete information. In a parallel effort, we have developed such a query language [12], and are examining ways of using it as an inference system behind an SQL-like language.

In section 4 we described a prototype implementation of our merging operation. While a considerable amount of work would be required to make this into a serious tool, it nevertheless goes some way towards demonstrating the feasibility of our approach. The merging operation itself was extremely easy to implement since it is so well specified; an efficient implementation took a matter of days to develop. Since the method can easily be applied to many other datamodels, it should also be straightforward to extend various existing database design tools with any merging facilities based on our method.

References

[1] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*. McGraw Hill, 1984.

[2] 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.

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

[4] 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.

[5] M. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, ch. Distributed Multidatabase Systems, pp. 425–456. Prentice Hall, 1991.

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

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

[8] P. Buneman, S. Davidson, and A. Kosky, "Theoretical Aspects of Schema Merging," To appear in *EDBT '92*. Available as a Technical Report, University of Pennsylvania.

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

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

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

[12] P. Buneman, S. Davidson, and A. Watters, "Federated Approximations for Heterogeneous Databases," in *A Newsletter of the Computer Society of IEEE*, pp. 27–34, August 1989.