Chapter 8
Mapping-Based Merging of Schemas

Rachel Pottinger

Abstract Merging schemas or other structured data occur in many different data models and applications, including merging ontologies, view integration, data integration, and computer supported collaborative work. This paper describes some of the key works in merging schemas and discusses some of the commonalities and differences.

1 Introduction

Schemas, ontologies and other related structures commonly need to be merged in a number of different applications. This happens for a number of reasons. For example:

View integration: Different users have their own aspects of a common application that they are interested in. For example, in creating a database for a university, the registrar has a different view from a professor, and both have different views from a student. In view integration, each user group creates its own "view" of what should be in the schema and then these different views are combined to create one global schema in which the data is stored.

Data integration: Users may want to query over multiple databases. For example, a BioMedical researcher may want to query both HUGO and OMIM for information on genes, and then use the gene information to query SwissProt for which proteins those genes encode. Because the researcher does not want to learn each of the schemas, and yet creating a warehouse of the entire set of databases is infeasible because of size and access restrictions, the user would like to just query one schema once and have the system figure out how to translate the queries over the sources. Such a system is called a data integration system.

R. Pottinger
University of British Columbia, 201-2366 Main Mall, Vancouver, BC, Canada V6T 1Z4
e-mail: rap@cs.ubc.ca

Z. Bellahsene et al. (eds.), *Schema Matching and Mapping*, Data-Centric Systems and Applications, DOI 10.1007/978-3-642-16518-4_8,

Merging ontologies: An ontology describes the concepts in a domain and the relationships between those concepts [Fikes 1996]. Ontologies are a commonplace in varied domains such as anatomy and civil engineering. Often a domain has more than one "standard" ontology for the same general concepts. For example, the foundational model of anatomy (FMA) [Rosse et al. 1998] is designed to model anatomy in great detail, whereas the Galen Common Reference Model [Rector et al. 1994] is designed to model anatomy for clinical applications. Because these two ontologies serve different communities, they have different concepts even though the domain is roughly the same. Merging the two ontologies would allow users to understand how all the concepts are related.

All of these applications have the same problem: given the two or more structured representations of data – which we often refer to as *models* [Bernstein et al. 2000] – combine the models to form one unified representation. These applications may also seek to create the mappings between the unified version and the input smaller schemas/ontologies. Many different works have looked at these different problems, both alone and in consort. This paper surveys some of the works in this area. In particular, Sect. 2 begins by describing a number of theoretical works that are relevant for multiple merging situations. Section 3 looks at works on view integration. Section 4 looks at work on data integration. Section 5 looks at work on merging ontologies. Section 6 looks at generic approaches for merging structured data representations. Section 7 surveys work on a variation of the problem: the data to be merged has been modified from a common ancestor and now the changes must be incorporated together. This variation is common both in file systems and in computer supportive collaborative work. Section 8 discusses commonalities and differences. Finally, Sect. 9 concludes.

Throughout this paper, we assume that the relationships between the schemas have already been created; this is beyond the scope of the paper. Interested readers in creating mappings are referred to existing surveys [Rahm and Bernstein 2001; Doan and Halevy 2004].

2 Theoretical Underpinnings

2.1 Information Capacity

The key notion of information capacity [Hull 1984] is that when comparing two schemas E and G, one can consider how much of the data in E can be accessed using G and vice versa.

Miller et al. [1993] study which properties of information capacity are required for both data integration and view integration. The key to understanding the requirements is the definitions of equivalence and dominance:

To be information capacity preserving, a mapping $\mathsf{I}(\mathsf{S1}) \rightarrow \mathsf{I}(\mathsf{S2})$ must be defined on every element in $\mathsf{S1}$ and functional in both directions. If so, then $\mathsf{S2}$ *dominates* $\mathsf{S1}$, denoted $\mathsf{S1} \leq \mathsf{S2}$. If $\mathsf{S1} \leq \mathsf{S2}$ and $\mathsf{S2} \leq \mathsf{S1}$, then $\mathsf{S1}$ and $\mathsf{S2}$ are equivalent,

denoted $S2 \equiv S1$, and I is an *equivalence preserving mapping*. Informally, this means that if S1 dominates S2, then it is possible to retrieve all the information from S2 by accessing S1; if the two are equivalent, one can get all the information from S2 by querying S1 and one can get all the information from S1 by querying S2.

Miller et al. show that in data integration, querying the source schemas from the integrated views requires that the integrated schema dominates the union of local schemas. In view integration, querying the integrated schema through the user views requires that the union of user views dominates the integrated schema. This notion of completeness in creating a merged or mediated schema is common, not just for information capacity but in other generic merging algorithms such as the specification by Buneman et al. [1992].

Ontology merging algorithms often use notions of completeness as well. However, ontology merging algorithms do not use information capacity as a basis for comparison since ontologies often lack data. Instead, they check to ensure that all concepts from the input ontologies appear in the merged ontology.

2.2 Instance-Level Constraints and Schema Merging

One natural question when examining work on merging schemas is how to deal with instance-level constraints such as key constraints and foreign keys. Unfortunately, as shown in Convent [1986] merging schemas is undecidable as soon as instance-level constraints are considered, even with a very simple representation of schemas. While Convent [1986] specifically considers relational view integration where the integrity constraints are keys and foreign keys, it generalizes to other schema merging areas as well.

Convent [1986] concentrates primarily on what it means to have incompatible constraints. Informally, this means that if users are trying to integrate views, then for each user's view, it should be possible to access those instances from the global schema – note that this is very similar to the information capacity requirement laid out in Sect. 2.1 by Miller et al. [1993]. Unfortunately, Convent [1986] shows that having incompatible constraints is undecidable even in this very basic case. Because of this early undecidability result, schema merging works typically do not consider instance-level constraints.

3 View Integration

As mentioned in Sect. 1, view integration is the problem of integrating the views/requirements that different users have of a schema, and then creating one global schema. Typically, this global schema is one in which the data is actually stored. Some systems may also allow the existing user views to persist, and then mappings may be created from the user views to the global schema where the data is stored.

This problem has been studied for quite some time, and is the subject of an early survey [Batini et al. 1986]. Batini et al. [1986] categorizes view integration work as taking one or more of the following steps:

Preintegration: Deciding which schemas to be integrated, in which order the integration should occur, and various preferences (e.g., if one of the schemas is "preferred" over the other).

Comparison of the schemas: Determining the correspondences and detecting the possible conflicts. In this context, a conflict is when a concept is represented differently in the input schemas. For example, a simple conflict might be that there is an attribute "Last Name" in one schema that is represented by an attribute "LName" in another schema.

Conforming the schemas: resolving the conflicts between the schemas; the authors note that automatic resolution is not typically possible in schema conformation.

Merging and restructuring: Now that the schemas are ready to be superimposed, how should they be combined? Batini et al. [1986] offers the following qualitative criteria to decide on the "correctness" of the merged schema:

- Completeness and correctness
- Minimality
- Understandability

These criteria are seen again and again in a number of different guises throughout the schema merging literature. As far as schema merging is concerned, this categorization is the main contribution of Batini, Lenzerini, and Navathe's paper; the bulk of the remainder is concentrated on matching. Again, matching (i.e., determining what concepts in one schema are related to the concepts in another schema) is outside the scope of this paper and is surveyed in existing surveys (e.g., Rahm and Bernstein 2001; Doan and Halevy 2004; Rahm 2011) (see also Chap. 2). Our work focuses on the "merging and restructuring."

The view integration problem was subsequently studied in many areas, including ER diagrams [Song et al. 1996; Lee and Ling 2003], XML [Beeri and Milo 1999; Tufte and Maier 2001; Yang et al. 2003], semi-structured data [Bergamaschi et al. 1999], relational and object-oriented databases [Larson et al. 1989; Shu et al. 1975; Biskup and Convent 1986; Navathe and Gadgil 1982; Shoval and Zohn 1991], and others. The remainder of this section details a few of the schema merging algorithms in the context of view integration.

3.1 Biskup and Convent

Biskup and Convent [1986] define a formal language for view integration and then proceed to integrate based on that language. This fairly early work provides a list of details need to be provided to create a view integration system:

- The data model.
- A language specifying the constraints of how the schemas are related. In this case, the authors use a variation on relational algebra. The precise constraints that are considered are described below.
- A formalization of conflicts. As in Batini, Lenzerini, and Navathe's work, a conflict is when a concept is represented differently in the input schemas (e.g., two corresponding attributes being called "Last Name" and "Lname").
- Some explanation of when the global schema will provide all the information that the users require (i.e., is it complete).
- A formal definition based on the concepts above.

After taking these items as input, the authors produce a global schema that meets those requirements, along with mappings that relate the global schema to the source schemas. The mappings between the views and the global schema are essentially the same as the language used to represent the constraints between the views for most of the constraints given. However, while the algorithm details what the global schema should be, and an example shows what the view to global schema mapping would look like, there is no algorithm given for creating the view to global schema mapping.

A key facet of their approach is that their constraints can only exist between single relationships. They also assume that each attribute can be related to a single other attribute. The set of constraints that they consider is entirely instancebased. These three restrictions combine to result in a fairly straightforward merge. Despite these limitations, the paper is a large step forward, largely to the overall framework. Informally, the constraints that they consider are:

- Identity constraints – the key of one relation is the same as the key of another relation, and their instances are the same.
- Selection constraints – the key of one relation is the same as the key of another relation. The instances of the key of one relation can be expressed as a selection of the instances of the key on the other relation.
- Disjoint constraints – the key of one relation is the same as the key of another relation. The instances of the keys are disjoint.
- Containment constraint – the key of one relation is the same as the key of another relation. The instance of one key is a subset of the instances of another relation. This relationship is like the selection constraint, but not easily quantifiable.

Unsurprisingly, given that one of the authors is also an author of the work on the undecidability of constraints in Sect. 2.2, they do not consider the case of conflicting constraints.

The desired result is a global schema, G, that fits two criteria, both based on Atzeni et al. [1982] notion of "weakly included." In particular, it must be that:

- G can be queried using only relational algebra and exactly the same view definitions will be retrieved.
- G is minimal – nothing that exists outside of the initial views is stored in G.

The first of these is similar to Hull's notion of maintaining information capacity [Hull 1984]; the second is one of the requirements in Batini et al. [1986].

Based on these requirements, the outcome of the merge is a set of relations, where each of the relations of the first three types of constraints (i.e., all but containment constraints) mean that the relations are combined, with all of their attributes present in the global schema. For containment constraints, the two relations are left separate, since there can be no way of determining what relational algebra operator to use.

3.2 Casanova and Vidal

Casanova and Vidal [1983] describe a method for performing view integration. In particular, they break the problem into two parts: combination and optimization.

The combination phase consists of gathering all of the views, as well as determining the dependencies between schemas. The optimization phase concentrates on both minimizing the schema and reducing redundancy. They concentrate on the optimization phase, which is akin to the problem considered in semantic merge.

The kinds of dependencies that they consider are, as in Biskup and Convent [1986], instance based. They consider the following dependency types:

- Functional dependencies
- An inclusion dependency says that one relation contains a subset of the tuples in another
- An exclusion dependency says that the instances of a relation are disjoint.
- A union functional dependency essentially allows functional dependencies to be declared valid across different schemas

The authors show that in general trying to optimize an integration with the above dependencies is intractable, so they concentrate on a limited subset of schemas. Essentially, these restrictions limit where the different kinds of dependencies can be applied, as well as assuming that the schemas are already in Boyce Codd Normal Form (BCNF).

Their goal is to create an optimization that is minimal in size, and also removes redundancy. They also want to ensure that any consistent state of the initial schema can be mapped into a consistent state of the transformed schema and vice versa – a notion very similar to the idea of completeness. To achieve these goals, their transformation removes many inclusion dependencies or union functional dependencies (since they may be a source of redundancy), as well as vacuous exclusion dependencies.

They provide an algorithm that will perform the optimization for the limited cases. Note that their algorithm creates the unified schema, but does not create the mapping from the unified schema to the initial views.

There are a number of other schema merging works in the view integration domain around this period, including Shoval and Zohn [1991] and Navathe and Gadgil [1982]. Generally, these works build on approximately the same foundation: define what it means to merge and what it means for there to be a conflict. Most

of these works assume that conflict resolution will be fully automatic. Additionally, most of these works assume that there is no *structural* conflict (e.g., that a merged column should belong to two different tables). This kind of more complex conflict resolution is explored more fully in the next section.

3.3 Spaccapietra and Parent

Spaccapietra and Parent [1994] work in the context of ERC+ [Spaccapietra et al. 1992], which extends an early version of entity relationship diagrams [Chen 1976]. In ERC+, there are three basic types of objects: attributes, entities, and relationships. While there are differences between ERC+ and Chen's original ER diagram, for the purposes of this paper, the primary difference is that ERC+ allows complex attributes (i.e., attributes can have subattributes) (Fig. 8.1).

Spaccapietra and Parent assume that the matching or aligning work has already been completed by the database administrator or some view integration tool. Their method for merging the data once these mappings (based on Superviews [Motro 1987]) have been created is fully automatic. However, this is at least partially possible because the authors do not consider constraints or ordering of any type, thus avoiding the undecidability result of Sect. 2.2.

Spaccapietra and Parent [1994] concentrates on rules for view integration, and the merging algorithm that uses them. They have two overriding principles: (1) integration rules should apply to both objects and the links between them, and (2) if there is a choice to be made, they choose the least restrictive option. According to these principles, they have created 6 integration rules that can be combined for their merging algorithm. The rules, as named by Spaccapietra and Parent, are:

1. *Elements integration rule:* For any two matching elements that are not of the same type, the resulting element is an entity type. If the two matching types are

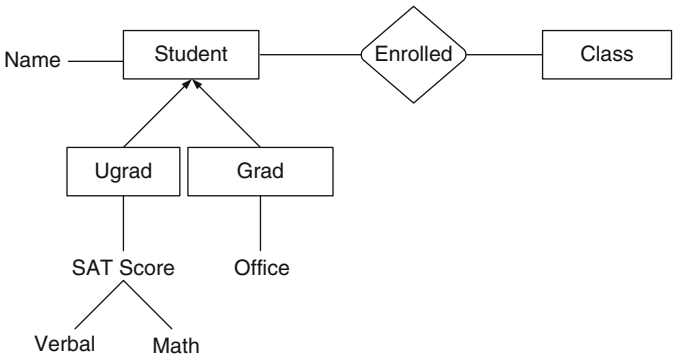

Fig. 8.1 An example ERC+ diagram. The entities are represented by *rectangles*, the relationships are represented by *diamonds*, and attributes are represented by plain text

not attributes and are of the same type, then the resulting element is of the original type.
2. *Links integration rule:* For any pair of elements that are matched to each other, if the elements are of the same type, then the links between them are of the same type.
3. *Paths integration rule:* If there is a direct path between two nonattribute elements in one model and an equivalent indirect path in the other model, choose the indirect path. The reason for this is that the indirect path contains more information, and the direct path can always be inferred from the indirect path.
4. *Integration of attributes of corresponding elements:* For elements that are already integrated, if each attribute pair in the original diagrams match, add the attributes to the resulting schema.
5. *Attributes with path integration rule:* If there is a direct path between an element and an attribute in one model and an indirect path between the corresponding pair in the other model, choose the indirect path.
6. *Add rule for elements and links without correspondent:* Add all elements that exist in one model and not in the other.

At a high level, their algorithm first merges the nonattribute elements, then merges the paths, and finally merges the attributes. The full algorithm is described in Spaccapietra and Parent [1994].

This work is notable because it marks a departure of focus from the existing literature: it concentrates on the fact that type information may conflict (rules 1 and 2) as well as the fact that some relationships that exist *explicitly* in the input schemas can be represented *implicitly* in the resulting schema (rules 3 and 5). This use of implicit relationships is featured prominently in later work, particularly in generic schema merging (Sect. 6). It is probably not a coincidence that this work, like the generic work, is not based in the relational model: the more direct representation of relationships between schema elements allows the authors to deal with complex relationships without running into the undecidability result in Convent [1986].

3.4 Rosenthal and Reiner

A contemporary work with Spaccapietra and Parent [1994] is The Database Design and Evaluation Workbench [Rosenthal and Reiner 1994]. The workbench allows users to manipulate schemas, including combining multiple views into one schema. Their ER-based system largely focuses on "rearrangements," which transform the input but leave the content "equivalent" to the input. The primary transformations that they considered were:

- Duplicating the attributes of an entity to propagate to related entities, or alternately removing duplication if necessary.
- Simplifying relationships by decomposing them into two simpler relationships.

- Inferring constraints from existing constraints, or alternately removing duplicate constraints.
- Creating keys.

The definition of equivalent, while different in a few details, is very similar to the notion of information capacity in Sect. 2.1. One scenario that they tackle is that of view integration. The authors state that their goals are much more pragmatic than some of the existing work; as previously discussed [Batini et al. 1986; Biskup and Convent 1986; Casanova and Vidal 1983], take a more theoretical approach. As such, Rosenthal and Reiner concentrate on a usable tool: they only detect homonyms and synonyms, and such conflicts are presented to the user for resolution. They then perform a duplicate removing union between the two schemas. No mappings are created between the schemas.

These works represent the type of focus on schema merging present in the more recent view integration literature. After this point, more of the database research in schema merging came from data integration (Sect. 4) and generic schema management (Sect. 6).

4 Data Integration

As motivated in the introduction, in data integration, there exists a set of heterogeneous, independent sources that contain related data. To have these sources be queried efficiently, a mediated schema can be created. Because these sources are independent, heterogeneous, and often change rapidly, it is not possible to then import all of the data into the mediated schema. Instead, the users query the mediated schema, and then the system translates the queries over the mediated schema into queries over the sources.

While much of the time this mediated schema is created manually, there exist a number of works that discuss creating the mediated schema based on the sources.

4.1 Data Warehousing

DWQ is a system for creating a data warehouse from the sources [Calvanese et al. 1999]. Calvanese et al. [2001] focuses on the data integration aspects of DWQ. This paper describes their system on how to use data integration in data warehousing. One issue with building a data warehouse is that it often has to be highly tuned for the specific queries; e.g., one might want to have a star schema (i.e., a base "fact" table from which various dimensions measuring things such as time and location radiate) for the data warehouse instead of whatever format just happens to be the merged result of the sources.

Their solution is to create a conceptual model, which "corresponds roughly to the notion of integrated conceptual schema in the traditional approaches to schema integration."

They consider that both the data warehouse and the source schema are views over the conceptual schema (i.e., local-as-view (LAV) [Vijayaraman et al. 1996]). As mappings they use "adorned queries," where the adornment is an annotation on the variables in the query; these are referred to as Reconciliation Correspondences. In particular, they consider three types of Reconciliation Correspondences: Conversion, Matching, and Merging Correspondences. Conversion Correspondences make data level transformations between the same real world objects. For example, one might use a Conversion Correspondence to translate between income defined monthly and income defined yearly. Matching Correspondences specify the matching. The Merging Correspondences show how to merge the data based on the existing Conversion and Matching Correspondences; they consist largely of calls to the Conversion and Matching Correspondences.

4.2 Pottinger and Bernstein

The authors of Pottinger and Bernstein [2008] take as input a pair of relational source schemas and a mapping between them, and then create a relational mediated schema and the mappings from the mediated schema to the source. They also show how this can be extended to a larger set of schemas. The mappings that they expect between the sources is a set of conjunctive mappings similar to the ones in Madhavan and Halevy [2003] – i.e., a set of select-project-join queries.

For example, assume that there are two travel websites: TravelOn and GoTravel. Assume that TravelOn has the relations TravelOnGuide(Name, ID) and TravelOn-Bio(ID, Bio) for tour guides and their bios, respectively. GoTravel may, in contrast, have the single relation GoTravel-Guide(Name, Bio) to represent both those concepts. One possible mapping between these sources is the following:

```
Guide(Name, Bio) :- TravelOn-Guide(Name, ID), TravelOn-Bio(ID, Bio)
Guide(Name, Bio) :- GoTravel-Guide(Name, Bio).
```

This mapping holds the standard Datalog semantics: Guides can be found either by taking the join of TravelOn-Guide and TravelOn-Bio on ID, or by looking at GoTravel-Guide. Hence, it shows that the two concepts are mapped to each other since instances of the same concept can be found by either conjunctive query. The question is: what should be in the mediated schema?

Informally, Pottinger and Bernstein [2008] requires completeness, accessibility to both all of the input relation (i.e., it preserves information capacity (see Sect. 2.1)), makes the concepts that are mentioned in the mappings accessible, does not combine relations unless they are related by the mappings, and finally is minimal.

In the case of our guide relation, this means that the mediated schema should contain the relation Guide(Name, ID, Bio). Additionally, there also needs to be mappings between the source schemas and the mediated schema. This is done through two sets of views. First, a set of views define an intermediate schema in terms of the mediated schema. These are called LAV views after the data integration architecture, where local sources are defined as views over the mediated schema [Vijayaraman et al. 1996]. Continuing with our travel example, the LAV views are:

```
I-TravelOn-Guide(Name, ID, Bio) :- Guide(Name, ID, Bio)
I-GoTravel-Guide(Name, Bio):- Guide(Name, ID, Bio).
```

A separate set of views defines the intermediate schema in terms of the sources. These are called global-as-view (or GAV) mappings after the data integration architecture, where the global sources are defined as views over the mediated schema (see [Lenzerini 2002] for a discussion of GAV as well as how it relates to LAV). Our final set of views for our travel example is thus the GAV views:

```
I-TravelOnGuide(Name, ID, Bio):- TravelOn-Guide(Name, ID),
                                 TravelOn-Bio(ID, Bio)
I-GoTravel-Guide(Name, Bio) :- GoTravel-Guide(Name, Bio).
```

An interesting result of this paper is that the mappings that are created between the mediated schema and the sources are a very limited form of global-local-as-view (GLAV) mappings [Friedman et al. 1999] (i.e., mappings where either the local or the global schema can be defined as the head of a view); in particular, the LAV views each only have one subgoal in them. This is important because the LAV views require using answering queries using views (see [Halevy 2001] for a survey), and having only one subgoal in the LAV view means that answering queries is very fast. This is particularly of note since the local sources will be related to each other in this fashion – regardless of how the mediated schema is created – so this result shows what we should expect even if the mediated schema is created in some other fashion.

4.3 BAV

Both-as-view (BAV) [Pidduck et al. 2002; McBrien and Poulovassilis 2003] is a framework to support schema transformation and integration. Similar to GLAV mappings [Friedman et al. 1999], BAV allows the definition of views between the mediated schema and the local sources in both direction – it treats both the global and the local schemas as sources. A key focus of their work is the transformations that make this possible – how can the mediated schema be related to the source schemas. They additionally provide a method to update a mediated schema based on the integration of new source schemas. To do so, they create a mapping that directly calls for the addition, deletion, and renaming of attributes and relations in the mediated schema.

5 Ontology Merging

An ontology specifies a domain-specific vocabulary of objects and a set of relationships between the objects in the vocabulary [Fikes 1996] (see also Chap. 2). In general, an ontology can be viewed as a graph of hierarchical objects that have specific attributes and constraints on those attributes and objects. The hierarchies in an ontology can be more than simple is-a generalizations and specializations – ontologies can also have notions of "containment" or "type of" or "part of." For the purposes of this paper, the different kinds of relationships do not matter; the fact that there are different kinds of relationships is the only part that is relevant. The objects in an ontology may have various structures; for the purpose of this paper, we can think of them as either being classes with various attributes or instances. While constraints may exist on attributes, classes, or instances, in general these constraints are not treated in the ontology merging literature.

Two sample ontologies are shown in Fig. 8.2. The classes are shown as ovals. The attributes are represented by text. A solid line represents inheritance. A dashed line indicates that an attribute is a member of that class. An example constraint might state that "SAT score" is at most 1,600.

5.1 SMART and Prompt

One representative work on merging ontologies began with an algorithm called SMART [Noy and Musen 1999a,b] and was later renamed Prompt [Noy and Musen 2000].

SMART tackles the problems of both merging and aligning ontologies. *Alignment* is the problem of taking two related ontologies and establishing links between them to indicate where they match (much as in the database literature). *Merging* is

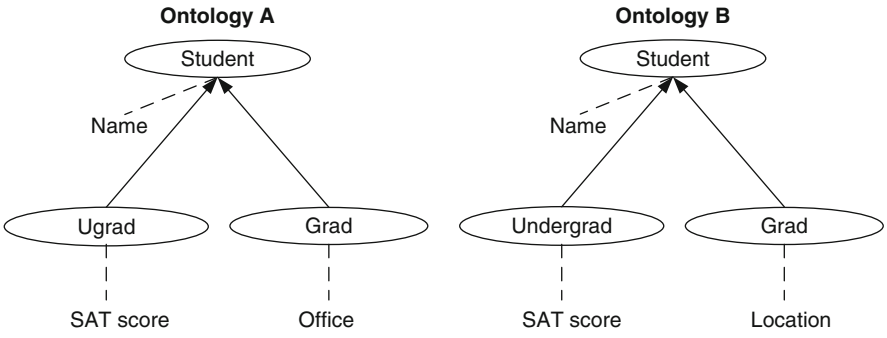

Fig. 8.2 Two sample ontologies. The classes are shown as *ovals*. The attributes are represented by *text*. A *solid line* represents inheritance. A *dashed line* represents that an attribute is a member of that class

the problem of taking two ontologies and combining them to form a third ontology. So, for example, alignment of the ontologies in Fig. 8.2 might tell us that element pairs with the same name are aligned with each other, that "Ugrad" matches with "Undergrad," and that "Office" matches with "Location." Merging the two ontologies would result in a single ontology with the same structure as both ontologies A and B since the two are structurally identical, but there would be some resolution of the naming differences (e.g., a decision would have to be made as to what to call the node that describes undergraduates).

SMART is used for both merging and alignment. The algorithm is not completely automatic for either operation; there are stages that must be performed by the user, even after the initial setup has occurred.

SMART keeps track of its state with two data structures: the *Conflicts list* and the *ToDo list*. The Conflicts list details the inconsistencies in the current state of the process that must be resolved before the resulting ontology can be in a "logically consistent state." The ToDo list keeps track of operations which should be done but are not required in order for the resulting ontology to be in a logically consistent state (e.g., if an action results in two attributes in the same classes with a similar name, SMART might add to the ToDo list a suggestion that one of them be removed). Since determining the source of a problem may enable the user to optimize the problem's resolution, each item in the ToDo and Conflicts list contains a reference back to the operation that triggered it. More details of SMART and Prompt, particularly on the matching and alignment aspects, can be found in Falconer and Noy [2011].

An outline of the SMART algorithm for merging is shown below. Note that the merging process requires also performing an alignment, so steps for both appear in the algorithm:

1. The user performs setup by loading the ontologies, A and B and specifying some options such as specifying if there is a preferred ontology.The result, the ontology C, is initialized to be a new ontology with a new root and A and B as that root's children.
2. SMART generates an initial list of suggestions of what should be aligned/merged. In this stage, SMART relies largely on content or syntactic information. The names of the objects are examined, but structural information (i.e., the position of the classes or their participation in specific relations) is not used.
   - For each pair of classes $a \in A$ and $b \in B$ with identical names SMART either merges the a and b in C or removes either a or b from C.
   - For each pair of classes $a \in A$ and $b \in B$ with linguistically similar names a link is created between them in C. This means that both a and b are still in C, but SMART suggests that they may need to be merged by adding them to the ToDo list.
3. The user selects and performs an operation such as merging a class or resolving an item on the ToDo or Conflict lists.
4. SMART performs any automatic updates that it can and create new suggestions. It has the ability to:

a. Execute any changes automatically determined as necessary by SMART.
b. Add any conflicts caused by the user's actions in step 3 to the Conflicts list.
c. Add to the ToDo list any other suggested operations or make new suggestions based on linguistic or structural similarity.

5. Steps 3 and 4 are repeated until the ontologies are completely merged or aligned.

5.2 *Chimæra*

The Ontolingua Server [Farquhar et al. 1996] is designed to make all parts of dealing with ontologies easier; they have a large collection of tools to allow users to create, view, manipulate, publish, and share ontologies. One of the tools is Chimæra[McGuinness et al. 2000], an environment for merging and testing ontologies.

Their system, like SMART, is designed to help users merge their ontologies. The difference is that where SMART concentrates on actually merging ontologies (e.g., automatically merging two classes with the same name), and Chimæra only points out the areas where merging is likely to be required. Their goal was to build a tool that "focuses the attention of the editor on particular portions of the ontology that are semantically interconnected and in need of repair or further merging." [McGuinness et al. 2000]

The authors identify a number of key features that a merging tool must support [McGuinness et al. 2000]. They propose that a merging tool have support for:

- Searching for names across multiple ontologies,
- Renaming in a systematic fashion,
- Merging multiple terms into a single term,
- Focusing the user's attention on term merging based on term names,
- Browsing classes and attributes,
- Modifying subsumption relationships in classes and attributes, and
- Recognizing logical inconsistencies introduced by merges and edits.

5.3 *FCA Merge*

FCA Merge [Nebel 2001] from Stumme and Maedche merges ontologies based on a lattice approach; they perform a match (in database terms) or an alignment (ontology terms) automatically. The lattice describes both the structure of the merged document and which elements in the ontology match according to the classes' semantic content. The created lattice may contain both nodes that are labeled with more than one class (indicating that merging may be required) and nodes with no corresponding class in the original ontology (suggesting that the user may want to insert a new

class). Interestingly, the lattices are found automatically [Bouzeghoub et al. 2000], but the merging is largely manual. To determine how to merge a node in the lattice, they distinguish four cases:

- *There is one class at the node:* In this case, the answer is found automatically; there are no conflicts and the class at that node is added to the resulting merged ontology.
- *There are two or more classes at the node:* In this case, the user is asked what should be done.
- *There are no classes at a nonroot node:* Here, the user must decide whether to add a class or not.
- *There are no classes at a root node:* In this final case, the user must decide whether to add a new top level class to the resulting ontology.

As seen from the description, FCA Merge makes no attempt to resolve any conflicts.

5.4 Ontology Merging Analysis

Each of the three systems, SMART, Chimæra, and FCA Merge, takes a very different approach to merging ontologies. Unlike database research (i.e., view integration and data integration), all three systems view the problem of merge to intrinsically require user intervention. SMART takes the most automatic approach of the three by merging some concepts from different ontologies without requiring any user intervention, but even this is limited: the user still must guide the system whenever the names of the classes that match are too different. Even if the names are linguistically similar, there is little that SMART can do automatically other than point the user at any potential conflicts unless the choice is clear from the preferred ontology. Chimæra provides very little automatic support; it focuses the user's attention on possibly related classes but has no conflict resolution support. FCA Merge provides amazing support for automatically matching the classes in the ontologies including doing some very sophisticated linguistic checks, but provides very little support for automatically merging classes in the ontology if any sort of conflict exists.

Together, these solutions define an overall compendium of interesting and useful features for ontology merging. SMART provides the notion of a preferred ontology that can help the system to work automatically. They also suggest the process of maintaining a list for the user of both where the user *must* perform an action and where the user *should* perform an action with the Conflict and ToDo lists. The Chimæra system offers good guidelines on what interactions must be available to merge ontologies. Finally, FCA Merge introduces the notion of additional nodes that are not present in either original ontology but may make the structure of the resulting ontology more sensible.

6 Generic Schema Merging

6.1 Buneman, Davidson, and Kosky

Buneman et al. [1992] delve into some of the theory of schema merging. In particular, they start once again with the assumption that elements are matched by name – i.e., they have avoided the matching problem entirely. The goal is to think of this from a lattice perspective (much like the one later used by FCA Merge) and describe two different theoretical perspectives: either the least upper bound (i.e., take everything that is available in either schema, which is rather like taking the union) or the greatest lower bound (i.e., take everything that is available in both schemas, which is rather like taking the intersection).

They, like most works here, focus on the least upper bound. Once having decided that the least upper bound is the correct semantics, the question is what kind of conflicts should be resolved. They use a very basic meta-model to allow them to concentrate on some very fundamental conflicts. In particular, their meta-model allows for elements which have only names as attributes. Their meta-model allows for two types of edges: is-a and has-a. They represent the fact that an element r Has-a element x of type y by an arrow from r to y with the label x. They do not consider constraints on the instances. Given these limited types of edges, they can focus on what would happen if two elements are combined resulting in the merged element having two types. For example, Fig. 8.3a says that element R has a Q of type P and S. Naturally, this does not make sense. Hence, their solution is to say that there should be a new type, and that both of the original types inherit from this type, as shown in Fig. 8.3b.

This kind of work shows the fundamental issues that have to be explored to merge schemas regardless of application or data model.

6.2 Model Management

Pottinger and Bernstein [2003] and Rondo [Melnik et al. 2003] both describe merge operators for Model Management [Bernstein et al. 2000]. The goal of Model

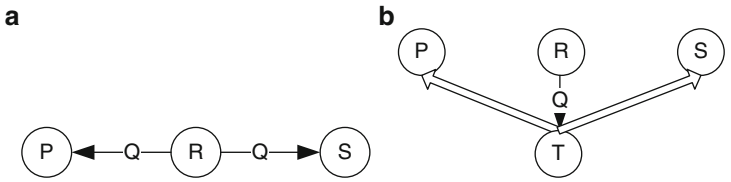

Fig. 8.3 Buneman et al. [1992] show that one conflict that occurs during the merging of schemas is that if there are two elements in the resulting merge that have different types (**a**), then a new type can be created which inherits from both original types (**b**)

Management is to allow structured representations of data – referred to as *models* – to be manipulated in bulk by using a set of generic and reusable operators. Because these works concentrate on the schema level rather than the data level, they do not consider instance level constraints.

Rondo [Melnik et al. 2003] is a Model Management system prototype. As such, it fully explores all Model Management operators (e.g., Merge, Match, Compose, Diff) and includes a Merge definition based entirely on equality mappings. Two elements can be declared to be equal, and each 1–1 mapping relationship can specify a preference for one element over another. Like Buneman et al. [1992], Rondo essentially creates the duplicate-free union of the elements and relationships involved. As with the view integration work in Sect. 3, both works consider schema-level conflicts, where elements in one schema are represented differently from elements in another schema (e.g., "Last Name" and "Lname"). Some conflicts require removing elements or relationships from the merged model (e.g., if an SQL column belongs to two tables in a merge result, it must be deleted from one of them). As in Pottinger and Bernstein [2003], Rondo's Merge resolves such meta-model conflicts later in a separate operator.

Pottinger and Bernstein [2003] concentrates on fully describing a Merge operator for Model Management. One if its contributions is defining three types of conflicts that have to be resolved in combining two schemas:

- *Representation conflicts:* Representation conflicts occur when there are two representations of the same real world concept. For example, the elements representing the concept of "name" may have different names. This corresponds to "comparison of the schemas" in Batini et al. [1986], and is resolved outside of Merge (since it may occur in other operators as well).
- *Meta-model conflicts:* Meta-model conflicts occur when the resulting Merge violates the constraints of a specific model, but not the constraints mandatory for *all* models. This is just like how in Rondo [Melnik et al. 2003] an SQL column can belong to only one table: there is nothing inherent in having structured data that says that a child must belong to only one parent. Similarly to Rondo, these conflicts must be resolved elsewhere.
- *Fundamental conflicts:* Fundamental conflicts are conflicts that occur during a Merge and must be resolved for the resulting model to be valid in *any* model. This notion of what must be true for any model is called the "meta-meta model" – for example, a relational schema is a model, the relational model is the meta-model, and the meta-meta model restricts what must be true in any meta-model.

Unlike many existing works, the Merge in Pottinger and Bernstein [2003] (hereafter Vanilla Merge after its meta-meta model, which is named Vanilla) allows complex mappings between elements in the model and many different types of relationships between elements; in particular, the mapping is also a first-class model. The mapping resolves conflicts by first taking values from the mapping, then from the (optional) preferred model, then from any model. For example, Fig. 8.4 shows two models (Model A and Model B) and a mapping ($\mathsf{Map}_{\mathrm{AB}}$) between them. This example extends an example in Pottinger and Bernstein [2003]. Figure 8.5 shows the

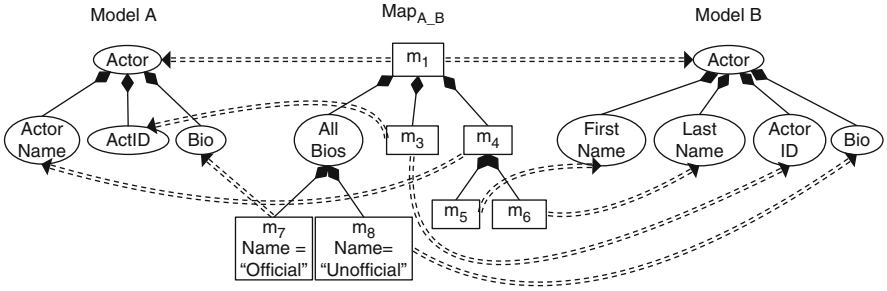

Fig. 8.4 An example mapping for the merge in Pottinger and Bernstein [2003]

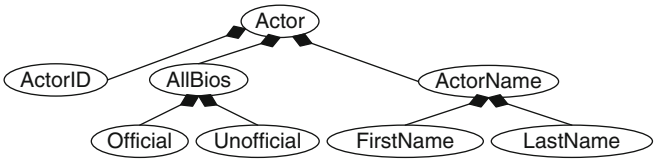

Fig. 8.5 The result of performing the merge in Pottinger and Bernstein [2003] on the models and mappings in Fig. 8.4

result of performing the merge in Fig. 8.4. The diamond-headed intra-model edges indicate a containment relationship. The double arrowed inter-model relationships indicate equality between elements. Looking carefully at $\mathsf{Map}_{\mathrm{AB}}$ reveals that there is an element All Bios, which is not mapped to by any element, but contains the Bios from models A and B, both of which have been renamed.

This complex mapping allows users to generically yet accurately represent the way that the elements are related to each other. As with other mappings, creating this style of mapping is beyond the scope of this paper; [Wang and Pottinger 2008] studies how to create these mappings.

Vanilla Merge defines what the output should be based on the principles laid out in many other papers referenced here, including the least upper bound notion from Buneman et al. [1992].

While Vanilla Merge defines that there may be other fundamental conflicts, the fundamental conflicts in the Vanilla meta-meta conflict are the one type conflict (an adjustment of the conflict discussed in Buneman et al. [1992] for the Vanilla meta-meta-model) and acyclicity.

6.3 Providing Choices to Users

Chiticariu et al. [2008] concentrate more on how the *user* interacts with the system. In particular, the authors assume that they are trying to merge a set of schemas. They model these schemas as graphs containing elements related by has-a relationships.

Like the model management work, they assume that they are given a set of input mappings that relate the two schemas. Unlike previous work on graph-based models, such as the ontology merging work in Sect. 5, the authors assume that if two elements are marked as being a match, that this is a *possible* place that they should be merged in the result. The goal of their system is to help users understand the space of possibilities that arise when a pair of elements are combined – what is the impact on the rest of the schema?

Radwan et al. [2009] build upon Chiticariu et al. [2008] by helping to automate the process of choosing how the schemas should be merged and presenting this result to the user. They use some of the schema matcher's internal information to help chose which possible elements to merge. A key feature of their system is that like Pottinger and Bernstein [2003] they create more complex edges than simple equalities. In particular, if two concepts are deemed similar, then they are either (1) merged, or (2) related through a "has" edge – they make this choice if the similarity between two elements is quite high in one direction but low in the other.

Both works provide a valuable complement to the more theory-based papers that make up the bulk of the papers cited here, and also dovetail nicely with some of the work on user preference in ontology merging systems, e.g., SMART (see Sect. 5.1).

7 Three-Way Merge

A final version of the merge problem is when the user is trying to merge two objects that are derivatives of a common ancestor. This problem occurs in both computer supported collaborative work (CSCW) [Munson and Dewan 1994; Berlage and Genau 1993; Berger et al. 1998] and file merging [Balasubramaniam and Pierce 1998]. In these contexts, there is a common ancestor and the two later models must be merged together based on the relationship between them and their common ancestor. With the added information from the common ancestor, the initial matching is much simpler, but the merging can be much more difficult.

For example, if there is a change that occurs only in one model, then the system will probably want to give preference to the changed model. This would seem to make the problem easier, but there are likely to be other constraints (e.g., the other model is preferred because it was modeled by an expert), which actually make the problem more difficult. Another example of this is that if both models diverge from the original, then it may be impossible to guess what the user wants. The presence of this information means that there are additional semantic constraints that must be satisfied, but the incompleteness of the information and the possibility of contradicting other information means that these additional constraints must be satisfied with no guarantee of a clear resolution to any of the constraints.

The work in this area that is the most flexible and automatic is that by Munson and Dewan [Munson and Dewan 1994]. They propose a system that, depending on the parameters that are specified, can be either manual, semiautomatic, or fully automatic. Users who choose a more automatic system are likely to receive a merged

result that does not correspond to exactly what they want, but they will not have to manually guide the system. The model that they look at, while applicable to files, considers the general case of objects by encapsulating the differences in the object types as described below.

They investigate two different types of merges, *consolidation merges* and *reconciliation merges*. In a consolidation merge, the assumption is that the changes made in both models should be integrated; most changes will not conflict with one another too much, and that changes from both models should be incorporated into their system. In a reconciliation merge, the assumption is that the merge is being performed to resolve conflicts. The two different types of merges call for very different outcomes.

Their system operates using a number of merge matrices. A merge matrix for two models to be merged, say A and B, has the rows represent edits made to achieve model A and the columns represent the edits needed to achieve model B. The matrix entries dictate how to merge the result of each (row, column) pair of edits. For example, a merge matrix for an object that is a sequence of elements may look like the one below (from Munson and Dewan 1994):

Sequence	Insert element	Delete element	Change element	No change
Insert element	Both			Row
Delete element		Row	User	Row
Change element		User	Merge edits	Row
No change	Column	Column	Column	

The blank entries represent situations that are either impossible (e.g., deleting an entry in one model and adding it in another) or where no change is required (e.g., if the element stays the same in both models). The entries that specify "row" mean that the version of the element from the model represented in the row should be taken, and similarly "column" indicates that the column's version should be taken. So, for example, if a sequence were unchanged in model A and deleted in model B, the action would be dictated by the entry at (no change, delete element), which, in this case is to delete the sequence since that is the action performed in model B, the one represented by the column.

The specification of these elements in the merge matrix is what allows for the algorithm to move on the scale from manual to automatic merging; if there are no entries that require manual intervention, then the algorithm is fully automatic.

In addition to the algorithm, they also list the characteristics that a merge tool should have. They state that a merge tool should support:

- Automatic merging,
- Interactive merging,
- Merging of both text files and more structured objects,
- Use of the semantics of the objects to drive the merging (rather than just a requirement that all objects merge in the same way), and
- Specification of preferred policies by the user.

8 Discussion

These diverse areas differ greatly in terms of goal, context, data model and more, but there are a number of similarities. We now discuss some of the similarities and differences across these different research areas. Because evaluating the quality of automatic merging algorithms is an open problem, we do not compare the approaches based on quality.

8.1 Separation of Match and Merge

Ontology merging, view integration, and CSCW all separate matching and merging. Ontology merging calls the difference alignment and merging [Noy and Musen 2000], but despite defining them as separate problems, the techniques that they use still force the system to perform both actions at the same time. The work on view integration also requires a matching step to occur before the merging process can begin. However, as it processes the matched elements, it may discover more matches that can be made to help with the merging process [Spaccapietra and Parent 1994]. CSCW and Model Management make a complete separation between matching and merging. A question is thus how much less efficient does it become to completely divorce the merging from the matching. In some cases, interleaving the matching with the merging (e.g., as in view integration) can cut down on the initial matching that needs to be done. However, there has been substantial work on schema matching as an independent problem (see [Rahm and Bernstein 2001; Doan and Halevy 2004] for some surveys). This increases the likelihood that future works on merging schemas will use the results of these matching algorithms as input and thus schema merging and matching will become more, rather than less, distinct over time.

The work in Radwan et al. [2009] represents a pull in the other direction, as it exploits information about the potential matches to suggest merge results. It would be interesting to see how the work in Radwan et al. [2009] can extend the work of various schema matchers.

8.2 Treating Models Asymmetrically

One idea that occurs repeatedly in all of these works is that the models are treated asymmetrically, allowing for the algorithms to function more automatically in the presence of conflicting choices. This may be because one model is more general or stable than the other and thus assumed to be the "preferred" model [Noy and Musen 1999a]. This allows merging operations to proceed much more automatically by giving it a clear indicator which element to prefer in the case of a conflict.

8.3 Data Model Differences

The different problems examined used very different data models (ontologies, ERC+ diagrams, etc), but for the most part the subset of the data models that were used were very similar. The authors largely ignored the parts of the data models that contained significant differences, (e.g., constraints in databases or ontologies).

All of the models that were considered were very simple; they essentially contained objects with subelements and these objects needed to be mapped; whether the subelement was a class or an attribute made very little difference.

The main difference that occurred was that in the merging of the ERC+ diagrams, the "relationship" elements had no real corresponding member in any of the other works. However, the relationships were treated in a fashion almost identical to those as entities, and when there was a choice to be made, the result was to revert to an entity.

The similarity of the approaches emphasizes the underlying common structure of the many diverse approaches.

8.4 Structure vs. Content

Another issue addressed by each system was how much the merge is driven by the structure vs. the content. Structure refers to the way that the objects relate to one another (e.g., class x is the parent of class y). Content refers to the data values for a given object (e.g., the name of the class is "Bob"). In most cases, the authors concentrated on the structure of models at least as much as the content, but both were necessary. An interesting point to note is that content is more likely to be needed in performing a match than in performing a merge, and thus systems that match and merge at the same time rely more on content than systems that make a cleaner separation between the two.

Of the ontology merging systems, SMART, on the one hand, concentrates mainly on resolving conflicts that arise by similar names, but it also looks to make sure that the children are merged. Chimæra, on the other hand, concentrates on structure over content. Two of its three modes of exploration rely entirely on structural searching and leave the content checking to the user. The third, however, relies completely on content. Both of these systems perform alignment at the same time as merging, however, so their dependence on content is not surprising.

In the third ontology merging system, FCA Merge, alignment is performed before the merge, and the merging pays very little attention to the content. In view integration, the problem is almost entirely structural if one considers paths to be entirely structural rather than content. The names are ignored almost completely as one would expect given that matching has already occurred.

8.5 Binary Merge vs. N-ary Merge

Many different situations, such as view integration, solve the problem of merging more than two models at the same time. However, in almost every case this is broken down into a series of two way merges. Break down of an n-ary merge follows either a ladder strategy, wherein a new component schema is integrated with an existing intermediate result at each step, or a balanced binary strategy wherein the models are divided into pairs at the start and are integrated in a symmetric pattern [Batini et al. 1986]. Thus even in situations where n-ary merge would be appropriate, the problem can be, and often is, broken down into a number of binary merges.

8.6 Can Merge be Fully Automatic?

One natural question is whether merge can be fully automatic. Based on these systems, it would appear that it cannot. Even with the limited problems that the above systems address, most of them require user intervention or error handling even though most do not consider either constraints or ordering.

However, with enough parameters set to allow the system to resolve any conflicts that arise, it is possible to have a largely if not entirely automatic version of merge. In addition to setting parameters to allow for the merge to be tailored to the above semantic differences, some of the parameters that should be considered are:

- Is there a preferred model and if so which one?
- If there are two matched objects and one has a more restrictive type than the other (e.g., an integer vs. a double), which would the user prefer? Both have their utilities; if there is an application built on top of the databases, it may be an advantage to require the more restrictive type. However, the less restrictive type allows more expressiveness.

A fully automatic system would not allow the different semantics and user desires to be fully taken into account. A system that required too many knobs would be annoying and frustrating for the user. Thus, any merge system must strike a balance. The most solution is to allow the setting of a number of parameters but to provide default settings to make the common case perform correctly.

8.7 User Interaction Requirements

Both the Chimæra ontology merger [McGuinness et al. 2000] and Munson and Dewan's work in CSCW [Munson and Dewan 1994] describe interactions that the user should be able to have with the tool when performing a merge. Combined, they list a great number of the goals that any merge operator for Model Management

should provide. The combined list yields that a generic merge operator should support:

- Renaming in a systematic fashion
- Merging multiple elements into a single element
- Modifying subsumption relationships in objects
- Automatic merging
- Interactive merging
- Merging many different types of objects
- The semantics of the objects to drive the merging rather than just requiring that all objects merge in the same way
- The users to specify what policies they prefer

Allowing all of these interactions will give the users the control that they need over the merging process.

9 Conclusions

This paper surveyed what it means to merge complex structures, such as relational schemas or ontologies, through a variety of applications, including view integration, data integration, and computer supported collaborative work. The work has a long history. Advances in other areas such as schema matching are likely to mean that work on merging continues to be a fruitful and interesting subject for the foreseeable future.

Acknowledgements Thanks are given to Phil Bernstein and Alon Halevy for their previous work and discussion with the author on the subject and to Jamila Salari, Steve Wolfman, and the editors for reading earlier drafts of this paper.

References

Atzeni P, Ausiello G, Batini C, Moscarini M (1982) Inclusion and equivalence between relational database schemata. Theor Comp Sci 19:267–285

Balasubramaniam S, Pierce BC (1998) What is a file synchronizer? In: ACM/IEEE international conference on mobile computing and networking (MOBICOM). pp 98–108

Batini C, Lenzerini M, Navathe S (1986) A comparative analysis of methodologies for database schema integration. ACM Comput Surveys 18(4):323–364

Beeri C, Milo T (1999) Schemas for integration and translation of structured and semi-structured data. In: International conference on database theory (ICDT). Springer, Heidelberg, pp 296–313

Bergamaschi S, Castano S, Vincini M (1999) Semantic integration of semistructured and structured data sources. SIGMOD Rec 28(1):54–59

Berger M, Schill A, Vöksen G (1998) Coordination technology for collaborative applications: Organizations, processes, and agents. Springer, London

Berlage T, Genau A (1993) A framework of shared applications with a replicated architecture. In: ACM symposium on user interface software and technology. ACM, NY, pp 249–257

Bernstein PA, Halevy AY, Pottinger R (2000) A vision of management of complex models. SIGMOD Rec 29(4):55–63

Biskup J, Convent B (1986) A formal view integration method. In: ACM SIGMOD international conference on management of data (SIGMOD). ACM, NY, pp 398–407

Bouzeghoub M, Klusch M, Nutt W, Sattler U (eds) (2000) Proceedings of the 7th international workshop on knowledge representation meets databases (KRDB 2000). CEUR Workshop Proceedings, vol. 29. CEUR-WS.org, Berlin, Germany, August 21, 2000

Buneman P, Davidson SB, Kosky A (1992) Theoretical aspects of schema merging. In: International conference on extending database technology (EDBT). Springer, London, pp 152–167

Calvanese D, Giacomo GD, Lenzerini M, Nardi D, Rosati R (1999) Data integration and reconciliation in data warehousing: Conceptual modeling and reasoning support. Network Inform Syst 2:413–432

Calvanese D, de Giomo G, Lenzerini M, Nardi D, Rosati R (2001) Data integration in data warehousing. Int J Cooper Inform Syst 10:237–271

Casanova M, Vidal V (1983) Towards a sound view integration methodology. In: PODS. ACM, NY, pp 36–47

Chen PP (1976) Entity relation model – toward a unified view of the data. ACM Trans Database Syst 1(1):9–36

Chiticariu L, Kolaitis P, Popa L (2008) Interactive generation of integrated schemas. In: SIGMOD. ACM, NY, pp 833–846

Convent B (1986) Unsolvable problems related to the view integration approach. In: ICDT. Springer, NY, pp 141–156

Doan A, Halevy AY (2004) Semantic integration research in the database community: A brief survey. AI Mag 25(1):109–112

Falconer SM, Noy N (2011) Interactive techniques to support ontology matching. In: Bellahsene Z, Bonifati A, Rahm E (eds) Schema matching and mapping. Data-Centric Systems and Applications. Springer, Heidelberg

Farquhar A, Fikes R, Rice J (1996) The ontolingua server: A tool for collaborative ontology construction. Technical Report KSL-96-26 KSL-96-26, Stanford University Knowledge Systems Laboratory

Fikes R (1996) Ontologies: What are they, and where's the research? In: Principles of knowledge representation and reasoning (KR), pp 652–653

Friedman M, Levy A, Millstein T (1999) Navigational plans for data integration. In: Proceedings of the national conference on artificial intelligence (AAAI). American Association for Artificial Intelligence, CA, pp 67–73

Halevy AY (2001) Answering queries using views: A survey. VLDB J 10(4):270–294

Hull R (1984) Relative information capacity of simple relational database schemata. In: Symposium on principles of database systems (PODS). ACM, NY, pp 97–109

Larson JA, Navathe SB, Elmasri R (1989) A theory of attribute equivalence in databases with application to schema integration. Trans Software Eng 15(4):449–463

Lee ML, Ling TW (2003) A methodology for structural conflict resolution in the integration of entity-relationship schemas. Knowl Inform Syst 5(2):225–247

Lenzerini M (2002) Data integration: A theoretical perspective. In: Symposium on principles of database systems (PODS). ACM, NY, pp 233–246

Madhavan J, Halevy AY (2003) Composing mappings among data sources. In: Very large data bases conference (VLDB). VLDB Endowment, pp 572–583

McBrien P, Poulovassilis A (2002) Schema evolution in heterogenous database architectures, a schema transformation approach. In: International conference on advanced information systems engineering (CAiSE), pp 484–499

McBrien P, Poulovassilis A (2003) Data integration by bi-directional schema transformation rules. In: International conference on data engineering (ICDE). Springer, London, pp 227–238

McGuinness DL, Fikes R, Rice J, Wilder S (2000) An environment for merging and testing large ontologies. In: Principles of knowledge representation and reasoning (KR), pp 483–493

Melnik S, Rahm E, Bernstein PA (2003) Rondo: A programming platform for generic model management. In: ACM SIGMOD international conference on management of data (SIGMOD). ACM, NY, pp 193–204

Miller RJ, Ioannidis YE, Ramakrishnan R (1993) The use of information capacity in schema integration and translation. In: Very large data bases conference (VLDB). Morgan Kaufmann, CA, pp 120–133

Motro A (1987) Superviews: Virtual integration of multiple databases. Trans Software Eng SE-13(7):785–798

Munson JP, Dewan P (1994) A flexible object merging framework. In: Conference on computer supported cooperative work (CSCW). ACM, NY, pp 231–242

Navathe SB, Gadgil SG (1982) A methodology for view integration in logical database design. In: VLDB. Morgan Kaufmann, CA, pp 142–164

Nebel B (ed) (2001) Proceedings of the seventeenth international joint conference on artificial intelligence, IJCAI 2001. Morgan Kaufmann, Seattle, Washington, USA, August 4–10, 2001

Noy NF, Musen MA (1999a) An algorithm for merging and aligning ontologies: automation and tool support. In: Proceedings of the Workshop on ontology management at sixteenth national conference on artificial intelligence (AAAI-99), Orlando, FL. Available as SMI technical report SMI-1999-0799

Noy NF, Musen MA (1999b) SMART: Automated support for ontology merging and alignment. In: Proceedings of the twelfth workshop on knowledge acquisition, modeling and management, Banff, Canada. Available as SMI technical report SMI-1999-0813

Noy NF, Musen MA (2000) Proceedings of the seventeenth national conference on artificial intelligence and twelfth conference on innovative applications of artificial intelligence. AAAI Press/The MIT Press, Austin, Texas, USA, July 30 – August 3, 2000

Pidduck AB, Mylopoulos J, Woo CC, Özsu MT (eds) (2002) Advanced information systems engineering, 14th international conference, CAiSE 2002, Toronto, Canada, May 27–31, 2002, Proceedings, Lecture Notes in Computer Science, vol. 2348, Springer, Heidelberg

Pottinger R, Bernstein PA (2008) Schema merging and mapping creation for relational sources. In: EDBT. ACM, NY, pp 73–84

Pottinger RA, Bernstein PA (2003) Merging models based on given correspondences. In: Very large data bases conference (VLDB). VLDB Endowment, pp 862–873

Radwan A, Popa L, Stanoi IR, Younis A (2009) Top k generation of integrated schemas based on directed and weighted correspondences. In: SIGMOD. ACM, NY, pp 641–654

Rahm E (2011) Schema matching and mapping. Bellahsene Z, Bonifati A, Rahm E (eds) Towards large-scale schema and ontology matching. Data-Centric Systems and Applications. Springer, Heidelberg

Rahm E, Bernstein PA (2001) A survey of approaches to automatic schema matching. VLDB J 10(4):334–350

Rector AL, Gangemi A, Galeazzi E, Glowinski AJ, Rossi-Mori A (1994) The GALEN CORE model schemata for anatomy: towards a re-usable application-independent model of medical concepts. In: Twelfth international congress of the European Federation for Medical Informatics, MIE-94, Lisbon, Portugal, pp. 229–233

Rosenthal A, Reiner D (1994) Tools and transformations – rigorous and otherwise – for practical database design. ACM Trans Database Syst 19(2):167–211

Rosse C, Shapiro LG, Brinkley JF (1998) The digital anatomist foundational model: principles for defining and structuring its concept domain. Proc AMIA Symp 1998:820–824

Shoval P, Zohn S (1991) Binary-relationship integration methodology. Data Knowl Eng 6:225–250

Shu NC, Housel BC, Lum VY (1975) Convert: A high level translation definition language for data conversion. Commun ACM 18(10):557–567

Song WW, Johannesson P, Bubenko J Janis A (1996) Semantic similarity relations in schema integration. Data Knowl Eng 19(1):65–97

Spaccapietra S, Parent C (1994) View integration: A step forward in solving structural conflicts. IEEE Trans Data Knowl Data Eng (TKDE) 6(2):258–274

Spaccapietra S, Parent C, Dupont Y (1992) Model independent assertions for integration of heterogeneous schemas. VLDB J 1(1):81–126

Tufte K, Maier D (2001) Aggregation and accumulation of xml data. IEEE Data Eng Bull 24:34–39

Vijayaraman TM, Buchmann AP, Mohan C, Sarda NL (eds) (1996) VLDB'96, Proceedings of 22th international conference on very large data bases. Morgan Kaufmann, Mumbai, September 3–6, 1996

Wang T, Pottinger R (2008) Semap: A generic mapping construction system. In: EDBT. ACM, NY, pp 97–108

Yang X, Lee ML, Ling TW (2003) Resolving structural conflicts in the integration of XML schemas: A semantic approach. In: ER. Springer, Heidelberg, pp 520–533