Model Management: First Steps and Beyond

Sergey Melnik
Microsoft Research, USA
melnik@microsoft.com

Abstract: Most information systems rely on meta data artifacts, such as database schemas, interface specifications, or view definitions, to store, transform, validate, and exchange information. Applications that produce and manipulate these meta data artifacts are complex and hard to build. The goal of model management research is to develop a set of powerful high-level operators that simplify the programming of such applications, and increase the productivity of developers by an order of magnitude. This paper discusses some results and challenges in generic model management based on a first published dissertation.

1 Introduction

Engineering and deployment of today's information systems involves a number of meta data manipulation tasks. To illustrate some of these tasks (in italics below), consider data integration. A key objective of data integration is to provide a uniform view of heterogeneous data sources. To construct a uniform view, source schemas are *matched* to identify their similarities and differences. The relevant portions of schemas are *extracted* and *integrated* into a uniform schema. The *translation* of data from the representation used at the sources into the representation conforming to the uniform schema is specified using database transformations, which may be expressed in SQL, XQuery, XSLT or other data manipulation languages. The queries that are stated against the uniform view are transparently *rewritten* into queries on sources. As the source schemas evolve, the database transformations and the uniform schema need to be *updated* accordingly.

The tasks resembling the ones above arise across a variety of other meta data intensive applications, which include data warehousing, data extraction, transformation, and exchange, and peer-to-peer data management. Despite these commonalities, applications that involve meta data manipulation tasks remain complex and hard to build, due to several reasons:

- Meta data applications are developed using low-level programming interfaces. Such interfaces typically provide access to the individual elements of meta data artifacts, such as individual attribute definitions of database schemas. The programming of meta data applications against such interfaces requires an extensive amount of navigational code and incurs high development and maintenance cost.
- Most approaches are application-specific. That is, reusing the code and infrastructure developed say for schema evolution to a data integration setting requires a major customization effort.
- The solutions are language-specific, i.e., are developed for SQL, UML, or XML, and are not easily portable to other domains. For example, solutions developed for

managing evolution of database schemas are hard to adapt to managing evolution of websites.

- No general-purpose platform is available to simplify the development of meta data tools and applications. The existing general-purpose solutions typically focus on persistent storage or graphical design environments for meta data artifacts and do not go far enough to support the developers of meta data applications. In fact, many of today's meta data related tasks are still solved manually, because an automated approach requires too much implementation effort due to the lack of a common programming platform.

To address these challenges, Bernstein et al. [BHP00, Ber03] outlined a vision to provide a truly generic and powerful environment to enable rapid development of meta data intensive applications in different domains. They called this capability *generic model management*. A *model* is a formal description of a meta data artifact. Examples of models include database schemas, ontologies, interface specifications, object diagrams, control flow diagrams, device models, and form definitions. The manipulation of models usually involves designing transformations between models. Formal descriptions of such transformations are called *mappings*. Examples of mappings are SQL views, XSL transformations, ontology articulations, mappings between class definitions and relational schemas, mappings between two versions of a model, mappings between device specifications and device functions, etc.

The key idea behind generic model management is to develop a set of algebraic operators that generalize the transformation operations utilized across various meta data applications. These operators are applied to models and mappings as a whole rather than to their individual elements, and simplify the programming of meta data applications. The operators are *generic*, i.e., they can be utilized for various problems and different kinds of meta data artifacts. Some of the major model management operators are:

- Match: semi-automatically create a mapping between two models.
- Merge: merge two models into a third one using a mapping between the two models.
- Extract / Diff: return a portion of a model that participates / does-not-participate in a mapping.
- Compose: return the composition of two mappings.

Model-management operators can be used for solving schema evolution, data integration, and other scenarios using short programs, or scripts, which are executed by a model management system. A high-level architecture of model management is depicted in Figure 1. The tools that deploy a model management system may maintain models and mappings in their own repositories, or may exploit the persistence capabilities of the model management system. The tools remain responsible for the management of model *instances*, such as data that resides in operational databases, XML documents, web pages, or device specifications, and may be capable of executing the mappings, i.e., transforming instances of one model into instances of another model.

If successful, generic model management may improve programmer productivity for meta data intensive applications by an order of magnitude. However, the vision for management of complex models raises many hard questions, such as the ones that were debated in the VLDB'00 panel [BHJ+00]:

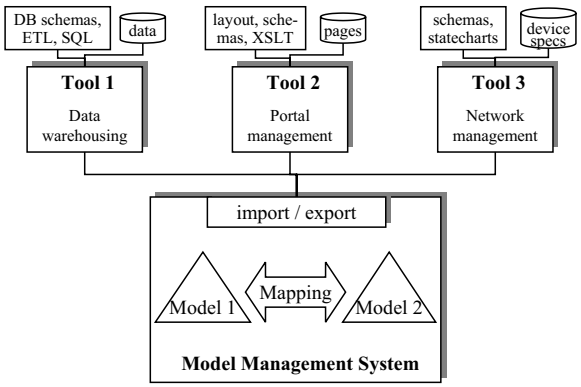

Figure 1: A high-level architecture of model management

- Is it feasible to develop a generic infrastructure for managing models and mappings? If so, what would it need to do, beyond what is offered in today's database management systems and repositories?
- Can we devise a useful generic notion of model that treats all popular information structures as specializations (SQL/XML/OO schemas, UML/ER diagrams, website maps, make scripts, etc.)?
- Can we produce a generic model manipulation algebra that generalizes transformation operations developed for data warehousing, integration, and translation?

One of the conclusions of the panel was that realizing the vision of generic model management would take years of research and that substantial implementation effort and theoretical work was required to answer the above questions to the full extent.

The questions raised in the panel set the stage for the first published dissertation on model management [Mel04]. Its objective was to demonstrate that model management operators are implementable and useful. The dissertation presents an initial study of the concepts and algorithms for generic model management. In the next sections, we highlight the key aspects of this work and discuss some subsequent efforts.

2 Change Propagation: a Ubiquitous Scenario

To illustrate the model management approach, consider a change propagation scenario. Initially, we have a "source" schema $_1$ whose instances are mapped to a "destination" schema d_1 by way of the mapping $_1\_d_1$ (see Figure 2). Subsequent changes of the source schema need to be propagated to the destination schema. This scenario surfaces in data exchange (where $_1$ is the schema of the operational database and d_1 is the exchange schema), data integration (where $_1$ is a local schema and d_1 is the integrated schema), data warehousing (where $_1$ and d_1 are the schemas of the warehouse and a data mart), and other settings. The mapping $_1\_d_1$ may be given as a view definition or, more generally, as a set of constraints that hold between $_1$ and d_1.

Suppose that $_1$ changes to $_2$. The change is described by the view $_1\_ _2$ that defines $_2$

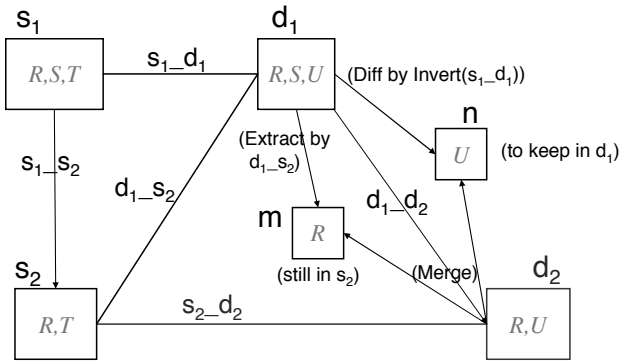

Figure 2: Propagating changes by the view $_1\_\ _2$

in terms of $_1$. Our goal is to obtain an updated version d_2 of d_1, and the mappings $d_1\_d_2$ and $_2\_d_2$ that relate the new versions of the schemas to the old ones. Mapping $d_1\_d_2$ tells us how to migrate the data from d_1 to d_2, while $_2\_d_2$ is the updated version of the input mapping $_1\_d_1$. Using the model-management operators, the solution can be obtained by executing the script shown below. The operators used in the script take models and mappings as input, and produce models and mappings as output; some operator invocations are nested:

script PropagateChangesByView($_1, d_1,\ _2,\ _1\_d_1,\ _1\_\ _2$)
1. $d_1\_\ _2 =$ Invert($_1\_d_1) \circ\ _1\_\ _2$;
2. $\langle\ \ d_1\_\ \ \rangle =$ Extract($d_1\ \ d_1\_\ _2$);
3. $\langle n\ \ d_1\_n\rangle =$ Diff($d_1\ $ Invert($_1\_d_1)$);
4. $\langle d_2\ \ d_2\_\ \ \ d_2\_n\rangle =$ Merge($\ \ n\ $ Invert($d_1_\ \ $) $\circ\ d_1\_n$);
5. $d_1\_d_2 = (d_1\_\ \ \circ$ Invert($d_2_\ \ $)) $\oplus$ ($d_1\_n \circ$ Invert(d_2_n));
6. $\ _2\_d_2 =$ Invert($d_1\_\ _2) \circ d_1\_d_2$;
7. return $\langle d_2\ \ _2\_d_2\ \ d_1\_d_2\rangle$;

To explain the intuition behind the operators and the above solution, assume that all schemas are relational, and that schema $_1$ contains the relations $R\ \ \ \ $, schema $_2$ contains $R\ \ \ U$ and so on as shown in Figure 2. Further, suppose that each mapping asserts that relations with equal names have equal extents, i.e., the mapping $_1\_d_1$ specifies that relations $R\ \ \ $ of $_1$ are copies of relations $R\ \ \ $ of $_2$, etc. Thus, to obtain d_2, we

- "extract" the relations of d_1 that are still connected to some relations in the modified schema $_2$, and obtain $\ \ $ as $\{R\} = \{R\ \ \ U\} \cap \{R\ \ \ \}$ (Line 2);
- compute the "difference" n between d_1 and $_1$ as $\{U\} = \{R\ \ \ U\} - \{R\ \ \ \ \ \}$ to determine the relations to be kept in the updated schema d_2 (Line 3); and finally,
- "merge" $\ \ $ and n to get d_2 (Line 4).

In our example, the script propagates the deletion of relation $\ \ $ from the source schema to the destination schema, as expected. The remaining lines of the script deal with producing

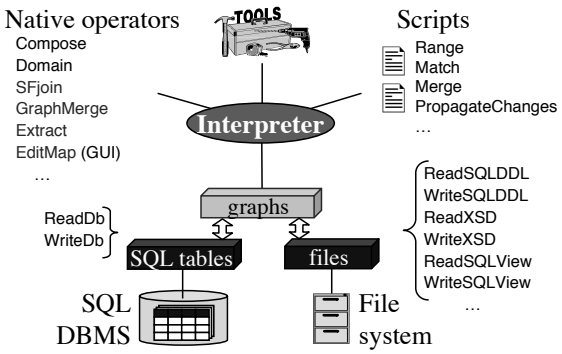

Figure 3: Architecture of the Rondo prototype

the correct mappings between the schemas using the operators Compose ($\circ$), Confluence ($\oplus$), and Invert.

Using the model-management operators, we were able to solve a challenging meta data management problem using seven lines of code, instead of the typical hundreds of lines. The solution script is language-independent, i.e., it factors out the specific schema and mapping languages. Thus, the solution developer can focus on the fundamental properties of the problem, and can reuse the solution across applications and domains.

3 Rondo: A Programming Platform for Model Management

To run the scripts such as the one presented in the previous section, we implemented the first prototype of a programming platform for model management, called Rondo [MRB03b]. The prototype supports the execution of model management scripts that are written using high-level operators, which manipulate models and mappings as first-class objects. In prior work, e.g., in [BHP00, BR00], detailed walkthroughs of various model-management problems have been examined to address the question of whether meta data management can be done in a generic fashion. Our contribution is that we succeeded in making such abstract programs executable. [1]

Rondo supports several schema languages, including relational and XML schemas, and simple mappings, called *morphisms*. Conceptually, a morphism is a set of arcs connecting the elements (e.g., relational tables, XML types) of two schemas. Clearly, a morphism is a weaker representation of a transformation between two models than an SQL view or a set of constraints. Morphisms are useful in metadata applications that do not require instance transformations, such as dependency tracking, model translation (e.g., UML to IDL or ER to SQL), and impact analysis. Furthermore, morphisms can represent mappings between different kinds of models (e.g., between a relational and XML schema), can always be inverted and composed, and can be implemented and manipulated easily.

Rondo implements all model-management operators suggested to date in the literature, and offers a graphical user interface for displaying and editing morphisms. Its architecture

[1] The source code and sample scripts of Rondo are available at http://www-db.stanford.edu/~modman/rondo/

is shown in Figure 3. The central component is an interpreter that executes scripts. The interpreter can be run from the command line or invoked programmatically by external applications and tools. Its main task is to orchestrate the data flow between the operators. The operators can be defined either by providing a native implementation, or by means of model-management scripts. Models and mappings are represented as structured objects in a common meta-meta-model and can be stored in a DBMS or file system. The operators are defined in terms of transformations of these structured objects.

In designing and implementing our prototype, we consciously focused on simplicity. We investigated how far we can go with a comparatively weak representation of models and mappings that can be used to solve an interesting class of problems. We also determined how much code is needed for a basic, but still useful, model management system. We showed that introducing a new model type like SQL DDL schemas in our prototype requires a moderate programming effort, but brings a large new class of model-management tasks within reach. The usefulness of the operators was studied in several model-management scenarios, such as change propagation and reintegration [MRB03a].

4 Match and Similarity Flooding Algorithm

Although many model-management tasks can be automated, there remain critical places where human decision-making is needed, e.g., to address the semantic heterogeneity. The operator Match, which establishes correspondences between models, inherently does not have formal semantics and is among the most difficult to automate. It gives us what we know about the relationship between models in the context of a particular application. Sometimes this relationship can be discovered semi-automatically [RB01] but ultimately Match depends on human feedback (and hence may be partial or even inaccurate). For example, it is likely that the input mapping $_1$—$_2$ from Section 2 can be obtained automatically with high accuracy if $_2$ is a minor modification of $_1$. In contrast, matching two independently developed schemas is closer to a computer-aided design task [BMPQ04].

In [MGMR02], we present an algorithm called Similarity Flooding (SF) that can be used for matching diverse data structures and is utilized for implementing the operator Match in the Rondo prototype. The input models are represented as directed labeled graphs and are used in an iterative fixpoint computation whose results tell us which nodes in one graph are similar to nodes in the second graph. For computing the similarities, we rely on the intuition that elements of two distinct models are similar when their adjacent elements are similar. Over a number of iterations, the initial similarity of any two nodes propagates through the graphs.

Usually, for every element in the matched models, the SF algorithm delivers a large set of match candidates. Hence, the immediate result of the fixpoint computation may still be too voluminous for many matching tasks. We examined several filters that can be used for choosing the best match candidates from the list of ranked matches returned by the SF algorithm. After filtering, a human is asked to verify and if necessary adjust the results. We suggested a novel accuracy metric that measures the quality of SF and other automatic schema matching algorithms by counting the number of the needed adjustments.

With help of the members of the Stanford Database Group, we conducted a user study to evaluate the effectiveness of our algorithm and to determine the fixpoint formula and the

filters that perform well. A discussion of the general issues with evaluating the performance of matching algorithms is presented in [DMR02].
We demonstrated the applicability of the SF algorithm for several matching tasks and examined its basic computational properties. Since its appearance in [MGMR02], variants and extensions of the algorithm were used for matching diverse artifacts including ontologies, web services, metabolic pathways, and synopses of medical images.

5 Operator Semantics: Leveraging Established Database Problems

The Rondo prototype was instrumental to clarify the intuition behind the model-management operators and to show that the operators are useful for solving practical problems. However, it does not tell the whole story, mainly because morphisms is a very limited mapping language. And yet, the purpose of many model-management scripts is to generate mappings that drive data migration, message translation, or database wrapping. Such mappings transform instances of models. How does a developer know that a script generates mappings that transform instances as expected? When designing a model-management system, how do we know that our operator implementation is correct? The answers require an understanding of the relationship between the models and mappings returned by each operator and the transformations expressed by those mappings on the states of those models. That is, a formal semantics for the operators is needed to explain what outputs should be produced if the input mappings are SQL views, XSL transformations, database constraints, etc.
Developing such semantics, which we call *state-based* semantics, is the subject of [Mel04, Part II]. We assume that a model denotes a set of states, or instances. For example, a relational schema denotes a set of database states; an XML schema denotes a set of XML documents; a workflow definition denotes a set of workflow instances; a programming interface denotes a set of implementations that conform to the interface. A mapping is a relation on instances. Thus, a mapping between two database schemas is a binary relation on the database states of these schemas, a mapping between XML schemas is a relation on XML documents, a data exchange mapping that generates XML messages from relational databases is a total function that assigns an XML document to each database state, etc.
The state-based operator semantics is defined by imposing constraints on the output models (sets of instances) and mappings (relations on instances). For example, the operator Compose ($\circ$) is defined as a set-theoretic composition of mappings:

$$_1\_\ _2 \circ\ _2\_\ _3 := \{(\ \ z) \mid \exists y((\ \ y) \in\ _1\_\ _2 \wedge (y\ z) \in\ _2\_\ _3\}$$

Operator Compose generalizes query composition and view unfolding. It is equivalent to view unfolding when $_1\_\ _2$ is a view on $_1$ and $_2\_\ _3$ is a query on $_2$, i.e., both are functional mappings. Other basic operators, such as Domain, Range, or Invert, have analogous natural definitions. To define the more complex operators, we exploit and generalize some other well-known problems studied in the database literature:

- The Extract operator builds on the problems of view selection in data warehousing [ACN00, CHS01, LBU01, TLS01], and finding exact rewritings for answering queries using views [CGLV05, Hal01]. Intuitively, to "extract" a portion of a schema

means to construct a schema that has the same query-answering capability as the original one for a given query workload. As a generalization of query workload, the operator takes as input an arbitrary mapping.

- The Merge operator is based on the desiderata put forth in schema and view integration [SP94, BDK92, Len02, AB01, PB03]. Intuitively, to "merge" two (view) schemas connected by a mapping means to construct a non-redundant schema that allows performing all queries and updates that can be done on the input (view) schemas. The operator covers a general case in which the input mapping may be not conflict-free, or inconsistent [BC86, LM98].
- The Diff operator exploits the view complement problem [BS81, LV03]. Two views are complementary if given the state of each view, there is a unique corresponding state of the source database. That is, if the two views are materialized then the database can be reconstructed from the views. Intuitively, "difference" is a portion of a model that complements the "extracted" portion, i.e., merging the two portions yields the original model. The operator covers the general case in which the input mapping is non-functional (i.e., not a view).

The problems mentioned above have typically been studied in isolation and trimmed to specific languages. We distill essential properties of these problems into language-independent operators, which can then be combined to address other problem settings, such as change propagation illustrated in Section 2. A number of other database problems can be set into the context of the model-management algebra:

- Finding a redundant peer in a peer-to-peer network amounts to determining whether a mapping is functional, i.e., the data stored at the redundant peer can be computed from the data stored at other peers.
- Determining whether schema integration constraints are conflict-free [BC86] amounts to testing whether the input mapping is a total and surjective relation.
- Testing query containment is a special case of testing mapping containment.

In [Mel04], we present detailed examples that substantiate and illustrate the state-based operator definitions using relational schemas and SQL views. We derive alternative formulations of operator definitions that are substantially easier to work with. We discuss the state-based semantics of the conceptual structures and operators used in Rondo.

Although state-based operator definitions, such as the above definition of Compose, look quite simple, implementing them is very challenging. Technically, computing the operator or script amounts to finding expressions in concrete model and mapping languages (e.g., SQL DDL or XQuery) that specify precisely the desired output models and mappings. In a recent effort, Fagin et al. [FKPT04] arrived independently at the same natural definition of mapping composition that we suggested, and embarked on a systematic study of the properties of the composition operator. They focused on relational schemas and a class of mappings called source-to-target tuple-generating dependencies (st-tgds), which are expressions of the kind $Q_1 \subseteq Q_2$, where Q_1 is a conjunctive query over the "source" schema and Q_2 is a conjunctive query over the "target" schema. Fagin et al. showed that the composition of two mappings given by st-tgds cannot always be given by a set of st-tgds, i.e., composition is *not closed* with respect to this class of mappings. To overcome this

problem, they introduced a second-order extension of st-tgds, which allow existentially quantified function symbols yet have an efficient chase procedure. Fagin et al. showed that second-order st-tgds are closed under composition and presented a composition algorithm whose output may be exponential in the size of the inputs. As one of the negative results, they established that composing first-order formulas is undecidable.

It is unlikely that implementing other model-management operators for model and mapping languages of practical importance is any easier.

6 Conclusions

Generic model management is a rich emerging area of research. Its ultimate goal is to build model-management systems that help solve challenging meta data management problems. We laid some stepping stones toward that goal by building a first running prototype, studying the usefulness of the model-management operators in practical scenarios, and examining the semantics of operators and scripts.

The research issues outlined in this paper are part of the agenda of the Model Management project[2] at Microsoft Research. In [BMPQ04], we discussed what it takes to build a robust schema matching platform. In another recent effort, we studied composition for a broader class of mapping languages and showed that the operators Domain, Range, and Compose are computationally equivalent, i.e., an algorithm developed for one of them can be used for computing the others. Other problems under investigation, which are motivated by feedback from product groups, include schema evolution, data migration, and the operator ModelGen, which translates models from one language into another (e.g., ER to SQL or SQL to XML).

Model management offers a wide spectrum of exciting open problems, which are technically challenging, motivated by real user requirements, and have plenty of research content. Some of these problems are rooted in the foundations of database theory, others are better solved using heuristic approaches. We embarked on [Mel04] thinking of model management as a project. Along the way, we uncovered so many hard problems that are beyond what could be accomplished in a single thesis that we now regard model management as a field. We expect that some of the problems can be solved by generalizing approaches that were developed for specialized settings. Others may be solved by applying techniques from related fields, such as programming languages, computer-aided design, mathematical logic, finite automata, etc. We expect they will keep many research groups busy for many years to come.

References

[AB01] Suad Alagic and Philip A. Bernstein. A Model Theory for Generic Schema Management. In *Proc. DBPL*, 2001.

[ACN00] Sanjay Agrawal, Surajit Chaudhuri, and Vivek Narasayya. Automated Selection of Materialized Views and Indexes in Microsoft SQL Server. In *Proc. VLDB*, 2000.

[BC86] Joachim Biskup and Bernhard Convent. A Formal View Integration Method. In *Proc. ACM SIGMOD*, pages 398–407, 1986.

[2] http://research.microsoft.com/db/ModelMgt/

[BDK92] Peter Buneman, Susan B. Davidson, and Anthony Kosky. Theoretical Aspects of Schema Merging. In *Proc. EDBT*, 1992.

[Ber03] Philip A. Bernstein. Applying Model Management to Classical Meta Data Problems. In *Proc. CIDR*, 2003.

[BHJ+00] Philip A. Bernstein, Laura M. Haas, Matthias Jarke, Erhard Rahm, and Gio Wiederhold. Panel: Is Generic Metadata Management Feasible? In *Proc. VLDB*, pages 660–662, 2000.

[BHP00] Philip A. Bernstein, Alon Y. Halevy, and Rachel Pottinger. A Vision of Management of Complex Models. *SIGMOD Record*, 29(4):55–63, 2000.

[BMPQ04] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. Industrial-Strength Schema Matching. *SIGMOD Record*, 33(4), 2004.

[BR00] Philip A. Bernstein and Erhard Rahm. Data Warehouse Scenarios for Model Management. In *Proc. ER*, pages 1–15, October 2000.

[BS81] François Bancilhon and Nicolas Spyratos. Update Semantics of Relational Views. *ACM TODS*, 6(4):557–575, 1981.

[CGLV05] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based Query Processing: On the Relationship between Rewriting, Answering, and Losslessness. In *Proc. ICDT*, 2005.

[CHS01] Rada Chirkova, Alon Y. Halevy, and Dan Suciu. A Formal Perspective on the View Selection Problem. In *Proc. VLDB*, pages 59–68, 2001.

[DMR02] Hai Hong Do, Sergey Melnik, and Erhard Rahm. Comparison of Schema Matching Evaluations. In *Proc. GI-Workshop Web and Databases, LNCS 2593, Springer, 2003*, 2002.

[FKPT04] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. Composing Schema Mappings: Second-order Dependencies to the Rescue. In *Proc. PODS*, 2004.

[Hal01] Alon Y. Halevy. Answering Queries Using Views: A Survey. *VLDB Journal*, 10(4):270–294, 2001.

[LBU01] Chen Li, Mayank Bawa, and Jeffrey D. Ullman. Minimizing View Sets without Loosing Query-Answering Power. In *Proc. ICDT*, pages 99–103, 2001.

[Len02] Maurizio Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. PODS*, pages 233–246, 2002.

[LM98] Jinxin Lin and Alberto O. Mendelzon. Merging Databases Under Constraints. *Intl. Journal of Cooperative Information Systems*, 7(1):55–76, 1998.

[LV03] Jens Lechtenbörger and Gottfried Vossen. On the Computation of Relational View Complements. *ACM TODS*, 28(2):175–208, 2003.

[Mel04] Sergey Melnik. *Generic Model Management: Concepts and Algorithms*. Ph.D. Thesis, University of Leipzig, Springer LNCS 2967, 2004.

[MGMR02] Sergey Melnik, Hector García-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *Proc. ICDE*, pages 117–128, 2002.

[MRB03a] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Developing Metadata-Intensive Applications with Rondo. *Intl. Journal on Web Semantics*, 1, 2003.

[MRB03b] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: A Programming Platform for Generic Model Management. In *Proc. ACM SIGMOD*, 2003.

[PB03] Rachel Pottinger and Philip A. Bernstein. Merging Models Based on Given Correspondences. In *Proc. VLDB*, 2003.

[RB01] Erhard Rahm and Philip A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 10(4):334–350, 2001.

[SP94] Stefano Spaccapietra and Christine Parent. View Integration: A Step Forward in Solving Structural Conflicts. *TKDE*, 6(2):258–274, 1994.

[TLS01] Dimitri Theodoratos, Spyros Ligoudistianos, and Timos K. Sellis. View Selection for Designing the Global Data Warehouse. *Data and Knowledge Engineering (DKE)*, 39(3):219–240, 2001.