



Generic Model Management

Zinovy Diskin¹

SWD Factory, Latvia and Universal Information Technology Consulting, USA

Boris Kadish

SWD Factory, Latvia

Generic model management (gMMt) is a novel view on classical and modern metadata management problems. The present article surveys the goals, components, pros and cons of gMMt, and major problems cited in the literature. It argues that some methodology developed in abstract mathematics can be extremely helpful for the field and is capable of providing it with a convenient notation, semantic foundations and truly generic specification patterns. The two other articles, titled *Mathematics of Generic Specifications for Model Management, I* (further referred to as Math-I, see p. 351), and *Mathematics of Generic Specifications for Model Management, II* (further referred to as Math-II, see p. 359), give some evidence to these claims by demonstrating how the machinery works in a series of examples.

WHY: FROM ELEMENT-AT-A-TIME TO MODEL-AT-A-TIME PROGRAMMING

Many data management routines include *metadata* applications that manipulate descriptions of data, usually called *schemas*, rather than the data itself. Typical examples are database design, schema integration and evolution, reverse engineering, data integration and translation, or data warehousing. Lately, the Web's dramatically rapid invasion into the field has added to this list several new tasks: ontology engineering and integration, Web site design, and XML wrappers generation. It has also multiplied the importance and diversity of versions of classical tasks by a big coefficient of e-commerce applications. In the OO jargon, data schemas (more generally, metadata artifacts) are often called *models*, so applications listed above can be classified as *model management* (MMt).

Along with models, MMt includes specifying and operating relations between models, which are usually called *model mappings* in the literature. Some examples are mappings between ER- or UML-diagrams on consecutive stages of design or between different releases of a database schema; mappings and their inverses between

ER-diagrams and SQL schemas implementing them; mappings between XML schemas to manage message translation; and various sorts of mappings between various UML diagrams either of one sort (*homogeneous*) or between different sorts (*heterogeneous* model transformation). The construct of view, well known in the relational data model, also presents nothing but a special syntax for specifying a mapping between relational schemas. In a sense, MMt is all about mappings.

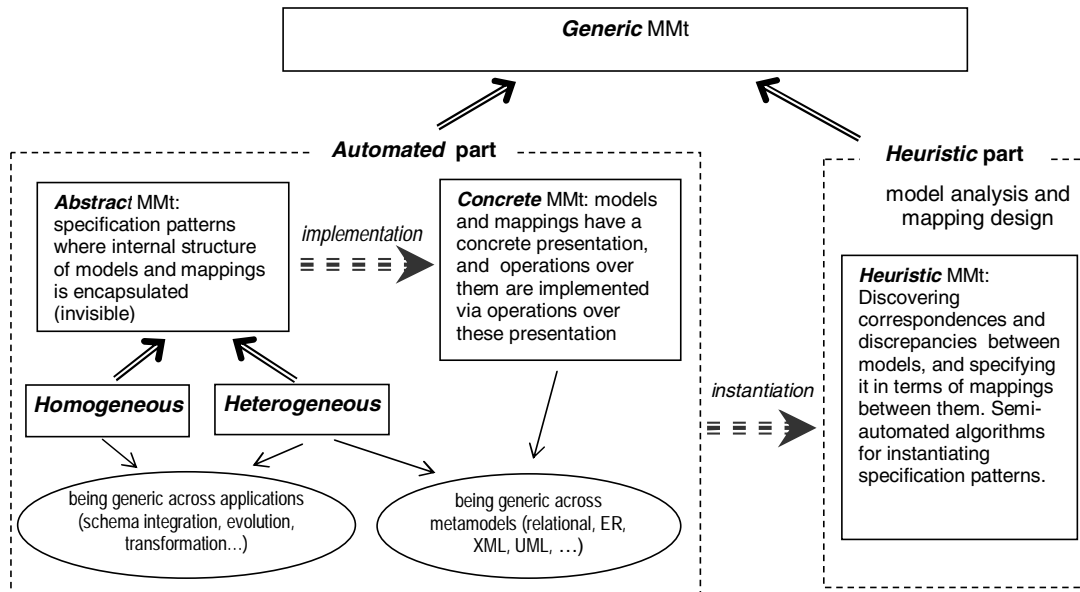
A commonly accepted standard approach to implementing MMt tasks is to present models and mappings as collections of objects and to program manipulations with them via programming manipulations with objects they consist of. Bernstein (2003) calls this *object-at-a-time* programming. A better term might be *element-at-a-time* programming, to emphasize working on the level of elements from which models and mappings are built. Though it does the job, element-wise programming is very laborious and error-prone. In a sense, it is similar to record-at-a-time programming in data processing. As is well known, eliminating the latter in modern DBMSs raised data processing technology on a qualitatively new level in programmers' productivity and semantic transparency.

Similarly, we can expect that a *model/mapping-at-a-time* programming environment, where the application programmer can think of MMt routines in terms of operations over models and mappings as integral entities, could essentially facilitate development and maintenance of metadata applications. To be really useful, such an environment should be *generic*, that is, be applicable to a wide range of MMt tasks involving a wide range of models of different types, i.e., of different *metamodels*. In this way we come to the idea of *generic MMt* (gMMt) environment manifested by Bernstein, Halevy, and Pottinger (2000).

WHAT: ABSTRACT, CONCRETE AND HEURISTIC PARTS OF GENERIC MMt

To achieve its goals, gMMt must resolve the following three major groups of problems, which are schematically presented in Figure 1.

Figure 1. Three parts of generic MMt



A. **Abstract gMMt—Genericness across applications:**

First of all, we need a generic way of specifying collections of models and mappings. Then we need to find a set of basic operations with models and mappings so that any practically important MMt procedure could be presented as a composition of basic operations. In other words, like we need data definition and manipulation language in data management, in MMt we need *model and mapping definition and manipulation language*. Table 1 compares MMt concepts to be developed with their analogs in the relational view to data management. We call this part of MMt *abstract* since here the internal structure of models and mappings is encapsulated, and they are treated as holistic abstract entities.

Abstract MMt can be divided into two parts, *homogeneous* and *heterogeneous*, dependant on whether the models we deal with are of the same or different types (metamodels). The most important issue in heterogeneous MMt is model translation, and the most difficult

problem in abstract MMt is how to specify it in a generic way (so that, for example, transformations of ER-diagram into an SQL schema and the latter into an XML DTD would be particular instances of the same specification pattern).

B. **Concrete gMMt—Genericness across metamodels:**

To implement abstract MMt patterns and operations, we need to have some concrete representation of models and mappings. Moreover, this representation should be universal with respect to data models (metamodels) so that such diverse models as relational schemas, XML DTDs, or various dialects of ER- and UML-diagrams would all be instances of the same universal format. The problem is evidently far from being easy.

There is a reasonable fear that even if we find such a universal representation *U*, encoding models/mappings of some *particular* metamodel in *U*-terms can be bulky and unwieldy. Then an MMt system’s genericness (as many other sorts of genericness in software products) will be an asset for tool builders rather than for tool users. A counterproposal might be to implement model definition

Table 1. What is to be done in abstract MMt (to be continued in Math-II, Table 2, p. 362)

	Elementary Units	Repository Structure	Elementary Query	A Complete Query Language
Data Management	Value	Set of relations (tables)	Relational operation	Relational algebra (calculus)
Model Management	Model, mapping	??	??	??

and manipulation languages for each metamodel of interest separately rather than strive for a generic-across-metamodels solution. With this approach, however, in addition to a set of particular MMT implementations I_1, \dots, I_n for each metamodel M_1, \dots, M_n , we will need to implement an even greater set of model translations I_{ij} , since many of the MMT tasks are themselves heterogeneous, for example, integration of heterogeneous data sources on the Web or navigating from ER-diagrams to SQL schemas and then to their XML presentations. Developing and maintaining such a system would be really laborious.

Thus, genericness across metamodels appears to be a necessary working property of MMT systems. To realize it, we must find a model representation format that would combine metamodel universality with clear and compact encoding of particular models.

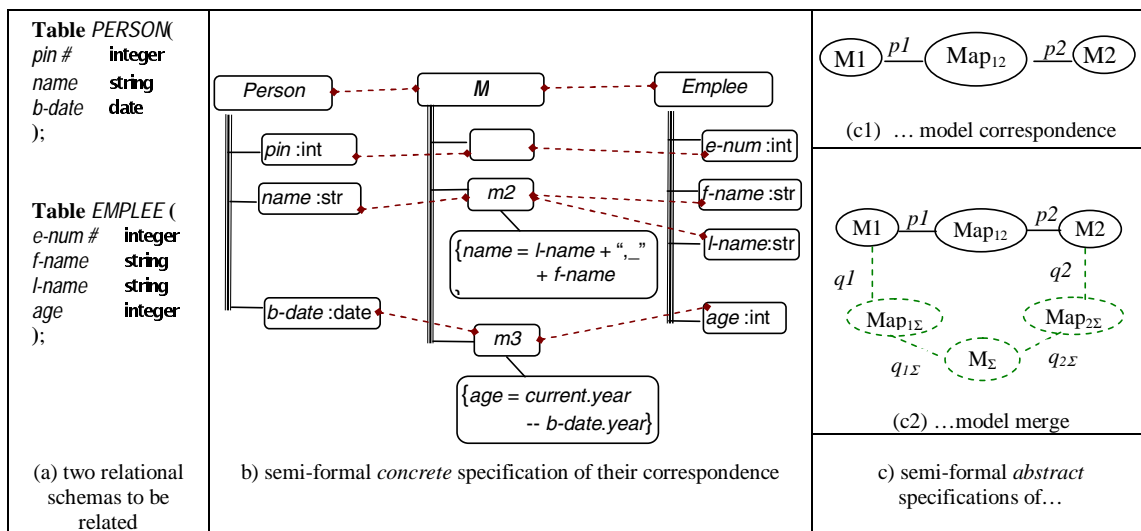
C. Heuristic gMMT—Discovering and specifying correspondences between models: As a rule, models involved in an MMT application present different views on the same data universe. Hence, we need to specify overlapping between models and present it as a model mapping. Here we face a highly nontrivial heuristic problem of *discovering* correspondences and discrepancies between models. Indeed, these models were often designed by different people in different contexts and for different goals. They may present the same data in different syntaxes, under different names, and in different formats. Particularly, the infamous problems of synonyms and homonyms in data schemas have to be resolved. In addition, data considered *basic* in one model can be *derived* in another model, that is, to be derivable/computable from the basic data of the other model.

In particular, data in one model can be metadata for another model. All these correspondences must be discovered (the first phase of heuristic MMT) and specified in a suitable format provided by abstract MMT theory (second phase). Of course, the first phase cannot be fully automated. Yet we may hope that some sophisticated algorithms based on special AI techniques (particularly machine learning) could assist the human in the essentially heuristic job of *model matching*.

To illustrate the concepts introduced above, we consider a very simple example in Figure 2. Two simple relational schemas are presented in Column (a) of Figure 2. Suppose we know that table *Person* with primary key column *pin* and table *Emplee* with primary key *e-num* refer to the same real-world objects. In addition, suppose that *Person.name* is nothing but concatenation of *Emplee.l-name* with *Emplee.f-name*. More formally, this description is presented in Column (b), where we use notation of Bernstein (2003) with minor distinctions. In detail, we have introduced a new model, whose each element is a relationship between the original models' elements: Elements *M*, *m1*, and *m3* are 1:1 and *m2* is a 1:2 relationship (of course, general M-N relationships are also possible). In addition, elements *m2* and *m3* have *expressions* attached, which state, correspondingly, that *Person.name* is concatenation of *Emplee.l-name* and *Emplee.f-name*, and *Emplee.age* is equal to the difference between the current year and *Person.bdate.year*.

Abstracting from this concrete specification, we come to the specification in Cell (c1) of Figure 2, where nodes denote models, and edges are relations between them.

Figure 2. Specifying correspondence between models (to be continued in Math-I, Figure 1, p. 352)



Each element of the abstract schema (c1) has an extent shown on the concrete schema (b): Extents of nodes are trees of elements, and extents of edges are sets of pairs (shown by dashed lines) of the corresponding elements.

Finally, Figure 2's Cell (c2) shows an abstract specification of a *model merge* operation: It takes the model diagram (c1) as its input and produces three more nodes with associated four edges:

$$(M_{\Sigma}, Map_{1\Sigma}, Map_{2\Sigma}) = \text{merge}(M1, M2, Map_{12})$$

The concrete “extent” of this operation is a procedure that takes the configuration in Column (b) as its input, manipulates with its elements by predefined rules, and produces a new merge model together with mappings to original models (this is not shown in Column (b)). We will return to this example in *Math-I* and consider it in more details and a better notation.

In this simple example, stating correspondence between models was an easy task. In real situations it may be a far more complicated problem. Finding systematic solutions to such sorts of problems is the essence of heuristic gMMt. Particularly, the heuristic operation **match** takes two models as input and produces the model diagram on Figure 2(c1). Then it becomes the input for **merge**.

All three parts of gMMt are equally important in making it a working environment capable of running practical applications. It is worth stressing, however, that abstract MMt is the specificational cornerstone of the entire building. It provides patterns that, on one hand, must be implemented in a concrete (yet generic) representation and, on the other hand, should be instantiated by data obtained from heuristic algorithms. Compact and adequate abstract MMt patterns would greatly ease the job of the other two MMt components.

A BIT OF HISTORY AND THE CURRENT SITUATION

MMt problems probably appeared on the stage simultaneously with the first DBMSs. As the technology matured, their weight in the problem basket of data management has been gradually increasing until the recent Web invasion into the field made data integration and transformation—a key MMt problem—a key issue for the entire field.

In each particular area of MMt problems, the database community accumulated a certain experience in both theory and practice. However, inventing a modeling language or writing an MMt application (though laborious) turned out to be much easier heuristically than extracting a common core of particular cases and

abstracting it into a generic framework spanning a wide class of languages and applications. Moreover, the very formulation of desired genericness in precise terms turned out to be a problem because of the absence of language to discuss and specify generic notions (cf. Drew, King, McLeod, Rusinkiewicz, & Silberschatz, 1993).

Nevertheless, under the pressure of practical needs and with a noticeable similarity amongst methods and tools addressing different MMt tasks, particular fragments of the main gMMt ideas have been gradually emerging and developing. Of primary importance among them was the regained interest in mappings between models. The latter first appeared in the literature probably in Paolini and Pelagatti (1977) and Kalinichenko (1978) and then were forgotten until the mid-'90s, when they reappeared on the stage in Diskin and Kadish (1997) and Diskin (1998). These papers proposed an integral formal framework (including a generic notion of query language) for what they called “data-model-independent” metamodeling, and the framework was essentially based on mappings (arrows) between models. Diskin (1999) presented an even more abstract version of the arrow framework and demonstrated its usefulness for clarifying meta-metamodeling issues and formalizing relations between modeling languages, particularly, sublanguages constituting UML. Independently, a detailed study of mappings in the context of the relational data model was undertaken in Miller, Haas, and Hernandez (2000) and Yan, Miller, Haas, and Fagin (2001).

The crucial step towards the formation of gMMt as an integral discipline was made by Bernstein et al. (2000), who manifested the issue and described a detailed agenda of what needed to be done. They once again stated the primary role of model mappings for generic MMt, introduced a set of basic operations with models and mappings, and demonstrated their use in a few application scenarios. Also, they proposed an object tree structure as a generic presentation of models. In fact, Bernstein et al. outlined all three parts of gMMt and initiated a few lines of research in the field. Finally, they coined the very term “generic MMt,” thus establishing a new discipline.²

To date, the following results have been achieved. An impressive progress has been made in heuristic MMt, where a few sophisticated algorithms for model matching were elaborated and a few methods developed; see Rahm and Bernstein (2001) for a survey and Halevy and Madhavan (2003) for a discussion of promising novel ideas in the field. A few tools based on semi-automated methods were implemented and tested in practical industrial projects (Madhavan, Bernstein, & Rahm, 2001; Mork & Bernstein, 2004; Velegarakis, Miller, & Popa, 2003; Yan et al., 2001). There has been a great progress in

Table 2. Problems of generic MMt by Bernstein (2003)

Machineries are to be built for:		
1. Filling the gap between models and mappings, which are syntactic structures, and their semantics, where models serve as templates for data instances and mappings govern translation of instances.	2. Representing models in a maximally generic yet tractable way.	3. A general-purpose interface between MMt operators and expression manipulation / inference engines (see Figure 2 for an example of expressions).
Clear and well-defined semantics is needed for:		
4. Operations of (a) Mapping Composition; (b) Model Merge; (c) Model Translation.	5. <i>Generic MMt</i> as a whole. Particularly, evaluation of <i>completeness</i> of a given set of basic operators is required. So far, the very notion of completeness of abstract MMt operations is not developed.	

concrete MMt as well. A convenient graph-based generic presentation for models was developed, and a first full implementation of a gMMt environment was built on its base (Melnik, Rahm, & Bernstein, 2003).

As for abstract MMt, a set of basic operations was further elaborated and applied to a few MMt application scenarios (Bernstein, 2003; Bernstein & Rahm, 2000). Analysis of mappings strongly modulated by the relational data model was continued in Madhavan, Bernstein, Domingos, and Halevy (2002); Velegrakis et al. (2003); and Fagin, Kolaitis, Miller, and Popa (2003). Also, a general categorical framework for MMt, focused on a generic pattern for specifying constraints in data models, was proposed in Alagic and Bernstein (2001).³ However, despite a few partial advances, abstract MMt remains the least developed part in gMMt. Due to its fundamental role for the entire program, unsolved abstract MMt problems essentially hinder gMMt's progress.

Problems to be Solved...

Bernstein (2003)—the most complete-to-date presentation of gMMt's goals and means—considers the problems presented in Table 2 to be the most pressing and evaluates the agenda of their solution as requiring “many years and many research groups to develop” (p. 220).

Such a cautious estimation is not surprising: The desired genericness of specifications on one hand and their graph-based (rather than string-based) nature on the other hand lead to an entirely new sort of specification problem. Indeed, so far major theoretical achievements in database research were related to one or another data model, and there are well-developed notions of query, constraint, and instance specialized to a particular data model; e.g., relational (rigorously formalized) or a particular dialect of ER or OO (much less clearly formalized though). Abstract generalization of these notions in a rigorous way presents a new sort of problem, where such

well-known and deserved mathematical means as first-order logic or relational algebra cannot help. Diskin, Kadish, and Piessens (1999) discuss the issue in detail.

... and Their Solutions: Category Theory vs. Generic MMt

Fortunately, software engineers and researchers do not need to develop specification foundations for gMMt from scratch. Mathematical *category theory* (CT) is a discipline specially devoted to specifying operations with complex objects in an abstract generic way independent of the objects' internal structure.

Categorically, everything that one wants to say about objects, one must say in terms of *morphisms* (or *arrows* or *mappings*) between objects. Such a reformulation is not evident and is sometimes really nontrivial, but the essential benefit is that the objects' internal structure does not occur in specifications. This allows CT to design really generic patterns (necessarily graph-based) for specifying and manipulating complex structures, and these patterns can be readily employed in gMMt.

There are three major categorical prescriptions that jointly form a framework where each of the problems in Table 2 can be successfully managed. Moreover, this framework eliminates the three major obstacles to realizing a truly generic MMt environment, which so far have been considered insurmountable (Bernstein, 2003):

1. Operation of model translation is inevitably metamodel specific.
2. Mechanisms for dealing with expressions also must be metamodel specific.
3. A compromise between expressiveness of model representation and tractability of operations over it is inevitable.

In this section, we will only name the three concepts

—Kleisly morphism, fibration and sketch—leaving their definitions and explanations of how they work for *Math-I* and *Math-II*.

Prescription 1 (for homogeneous abstract MMt)—**Consider model mappings as Kleisly morphisms:** This prescription sets up a proper framework for working with derived data and managing Problems 3 and 4(a, b) and a good part of Problem 5. In fact, Problems 3 and 4(a) are eliminated by reducing them to composition of Kleisly morphisms: Since Kleisly morphisms are functional mappings, their composition is easy and presents a form of term substitution. Kleisly morphisms put MMt on truly arrow (hence, generic) foundations and serve as a main vehicle of MMt’s categorization (see *Math-I* for details). Particularly, their convenience for specifying schema repositories, including relations between schemas (such as views and refinement) and operations over them (like schema integration), was shown in Kadish and Diskin (1996), Diskin and Kadish (1997), and Diskin (1998).

Prescription 2 (for heterogeneous abstract MMt)—**In a heterogeneous environment, consider data as a fibration of the data instances over the universe of schemas, in its turn fibred over the universe of metaschemas:** To realize this prescription, we need to choose some generic representation for data and schemas, for example, graphs. Then the fibrational view on data prescribes to supply (label) each item (an object identifier or a reference) in the data graph D with its type and organize the latter into a graph—the schema of the data S .⁴ Then, data can be seen as a graph mapping $\delta: D \rightarrow S$. Being applied to the metalevel, the fibrational view suggests that in a heterogeneous universe, we need to keep each model with its metamodel and label each item of the model with its type from the metamodel. This way we come to a mapping $\sigma: S \rightarrow M$ with M a graph specifying the metamodel. The fibrational view on data and metadata allows managing Problems 1 and 4(c) and, jointly with Prescription 1, Problem 5.

Following Prescriptions 1 and 2 we can build a specification framework where any reasonable MMt routine, including data and schema translation, integration, and evolution, is presented as a composition of a few basic diagram operations with data, schemas, metaschemas, and mappings between them. Definitions of these operations do not anyhow depend on the internal structure of data, schemas, and mappings. The basics of the approach are presented in *Math-I* and *II*.

Prescription 3 (for concrete MMt)—**Consider models as (visual presentations of categorical) sketches and data as sketch instances in the category of sets and mappings⁵:** It is mathematically proven that any data that can be specified formally can be specified by a sketch as well. Hence, the sketch format gives a really universal specification language of absolute expressiveness. Gen-

erally speaking, the sketch presentation of models is not a must for gMMt, but it greatly simplifies many issues by providing specifications combining the following advantages:

- **Extreme compactness:** A sketch consists of only three sorts of items—nodes, arrows, and diagram predicate declarations.
- **Clear and simple semantics:** For data modeling, nodes are sets, and arrows are mappings between them.
- **Clear and simple formal definitions of model mapping:** This property is especially valid for MMt.

Being applied to the metalevel, where models are data whose schemas are metamodels, Prescription 3 suggests to consider models as instances of sketches specifying metamodels. It allows us to apply many results of categorical logic to MMt problems. Details about sketches and their applications to data modeling can be found in Diskin (2003); Diskin and Kadish (2003); and Diskin, Kadish, Piessens, and Johnson (2000).

CONCLUSION

Generic MMt is an actively developing novel research field. Its creation has been motivated by purely practical reasons of effective metadata management. Nevertheless, gMMt quickly came to specification issues requiring fairly abstract mathematical means. Fortunately, the latter are already developed in category theory and can be readily adapted for MMt needs.

After a specification framework for gMMt is developed, the following two major tasks are in order. The first is to further develop a theory and algorithms for schema matching and to test them in various practical situations. It is a large and practically unlimited research and industrial field in-between DB and AI. The second major task is to build effective implementations of the gMMt environment and to estimate their efficiency in a variety of contexts and practical applications. Together these tasks form a broad experimental agenda still in its infancy. However, most likely it will rapidly mature and progress under the pressure of such important applications as data warehousing and the Web.

REFERENCES

Alagic, S., & Bernstein, P. (2001). A model theory for generic schema management. In *Eighth International Workshop on Databases and Programming Languages*,

DBPL'2001 (pp. 228-246).

Bernstein, P. (2003). Applying model management to classical metadata problems. In *International Conference on Innovative Database Research, CIDR'2003* (pp. 209-220).

Bernstein, P., Halevy, A., & Pottinger, R. (2000). A vision for management of complex models. *SIGMOD Record*, 29(4), 55-63.

Bernstein, P., & Rahm, E. (2000). Data warehouse scenarios for model management. In *International Conference on Entity-Relationship Modeling, ER'2000* (pp. 1-15).

Cadish, B., & Diskin, Z. (1996). Heterogeneous view integration via sketches and equations. In *Lecture notes in artificial intelligence: Vol. 1079. Foundations of Intelligent Systems: Ninth International Symposium, ISMIS'96* (pp. 603-612). Springer.

Diskin, Z. (1998). The arrow logic of meta-specifications: A formalized graph-based framework for structuring schema repositories (Tech. Rep. No. TUM-19820). In H. Kilov, B. Rumpe, & I. Simmonds (Eds.), *Seventh OOPSLA Workshop on Behavioral Semantics of OO Business and System Specifications*. Technische Universitaet Muenchen.

Diskin, Z. (1999). Abstract metamodeling, I: How to reason about meta- and metamodeling in a formal way. In K. Baclawski, H. Kilov, A. Thalassinidis, & K. Tyson (Eds.), *Eighth OOPSLA Workshop on Behavioral Semantics* (pp. 32-48) Northeastern University.

Diskin, Z. (2003). Mathematics of UML: Making the odysseys of UML less dramatic. In K. Baclawski, & H. Kilov (Eds.), *Practical foundations of business system specifications* (pp. 145-178). Kluwer.

Diskin, Z., & Kadish, B. (1997). A graphical yet formalized framework for specifying view systems. In *Advances in Databases and Information Systems, ADBIS'97: Vol. 1. Regular papers* (pp. 123-132)., St. Petersburg, Russia: Nevsky Dialect.

Diskin, Z., & Kadish, B. (2003). Variable set semantics for keyed generalized sketches: Formal semantics for object identity and abstract syntax for conceptual modeling. *Data & Knowledge Engineering*, 47, 1-59.

Diskin, Z., Kadish, B., & Piessens, F. (1999). What vs. how of visual modeling: The arrow logic of graphic notations. In H. Kilov, B. Rumpe, & I. Simmonds (Eds.), *Behavioral specifications in businesses and systems* (pp. 27-44). Kluwer.

Diskin, Z., Kadish, B., Piessens, F., & Johnson, M. (2000). Universal arrow foundations for visual modeling. In *Lecture notes in artificial intelligence: Vol. 1889. International Conference on the Theory and Applications of Diagrams* (pp. 345-360). Springer.

Drew, P., King, R., McLeod, D., Rusinkiewicz, M., & Silberschatz, A. (1993). Report on the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, 22(3).

Fagin, R., Kolaitis, P., Miller, R., & Popa, L. (2003). Data exchange: Semantics and query answering. In *International Conference on Database Theory, ICDT'2003* (pp. 207-224).

Goguen, J., & Burstall, R. (1992). Institutions: Abstract model theory for specification and programming. *Journal of ACM*, 39(1), 95-146.

Halevy, A., & Madhavan, J. (2003). Corpus-based knowledge representation. In *International Joint Conference on Artificial Intelligence, IJCAI'03*.

Kalinichenko, L. (1978). Data model transformation method based on axiomatic data model extension. In *International Conference on Very Large Data Bases, VLDB'78* (pp. 549-555).

Madhavan, J., Bernstein, P., Domingos, P., & Halevy, A. (2002). Representing and reasoning about mappings between domain models. In *18th Conference of American Association for Artificial Intelligence, AAAI'2002*.

Madhavan, J., Bernstein, P., & Rahm, E. (2001). Generic schema matching with Cupid. In *International Conference on Very Large Databases, VLDB'2001*.

Melnik, S., Rahm, E., & Bernstein, P. (2003). Developing metadata-intensive applications with Rondo. *Journal of Web Semantics*, 1, 47-74.

Miller, R., Haas, L., & Hernandez, M. (2000). Schema mapping as query discovery. In *International Conference On Very Large Databases, VLDB'2000*.

Mork, P., & Bernstein, P. (2004). Adapting a generic match algorithm to align ontologies of human anatomy. In *International Conference on Data Engineering, ICDE'2004* (pp. 787-790).

Paolini, P., & Pelagatti, G. (1977). Formal definition of mapping in a database. In *ACM SIGMOD Conference on Management of Data* (pp. 40-46).

Rahm, E., & Bernstein, P. (2001). A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 334-350.

Velegrakis, Y., Miller, R., & Popa, L. (2003). Mapping adaptation under evolving schemas. In *International Conference on Very Large Databases, VLDB'2003*.

Yan, L.-L., Miller, R., Haas, L., & Fagin, R. (2001). Data-driven understanding and refinement of schema mappings. In *ACM SIGMOD Conference on Management of Data*.

KEY TERMS

Data Model: A specification language where we can model and talk about models of a given class and their instances, manipulations with instances and models, queries, and constraints. A typical example of a well-developed data model is the relational model. ER data model, though less formal, is another example.

Data Transformation/Translation: Transformation of data stored in one format (metamodel) to another format.

Generic Model Management (gMMt): An MMt environment/system applicable to a wide range of MMt tasks across a wide range of metamodels.

Instance (of a Model): A set of data items structured according to the model. Data *instantiate* the model (or data schema in this context).

Metadata: Data specifying data, their structure, and presentation.

Metamodel (of a Given Class of Models): A model whose instances are the models of the class. Models of a given metamodel are called *similar*.

Model: A metadata artifact such as relational or XML schema, interface definition, or Web site layout.

Model Mapping or Morphism: Informally, a correspondence between models. Formally, it is a function (arrow) sending elements of the source model to elements of the target model. The graph of this function can be reified as a special model, often also called model mapping in the literature. We prefer the term *correspondence model* for reified mappings and use *model mapping* in the narrow sense of mappings-as-functions.

ENDNOTES

¹ Partially supported by Grant 05.1526 from the Latvian Council of Science.

² Inventing a good term is not just linguistic sugar; rather, it is an act of stating a notion or concept, and it may significantly speed up the progress in the field.

³ The framework is based on the notion of *institution* (Goguen & Burstall, 1992), which was invented for “model management” in mathematical logic (where models are logical theories). A discussion of institutions’ applicability to MMt problems can be found in Diskin (1999), where a bit more general version of institutions was developed for precise formulation of meta-metamodeling concepts.

⁴ This is nothing but the XML view on data, and indeed an XML document can be viewed as a fibration of data over tags, as soon as we arrange data and the set of tags into graphs. In a sense, XML’s authors invented fibrations independently of category theorists.

⁵ Sketch is a graph-based format used in category theory for specifying mathematical structures (see *Math-I* for details).