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.


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.

Copyright © 2006, Idea Group Inc., distributing in print or electronic forms without written permission of IGI is prohibited.
Generic Model Management

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)

<table>
<thead>
<tr>
<th>Data Management</th>
<th>Elementary Units</th>
<th>Repository Structure</th>
<th>Elementary Query</th>
<th>A Complete Query Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Set of relations (tables)</td>
<td>Relational operation</td>
<td>Relational algebra (calculus)</td>
<td></td>
</tr>
<tr>
<td>Model Management</td>
<td>Model, mapping</td>
<td>??</td>
<td>??</td>
<td>??</td>
</tr>
</tbody>
</table>
Related Content

A Measurement Ontology Generalizable for Emerging Domain Applications on the Semantic Web
[www.igi-global.com/article/measurement-ontology-generalizable-emerging-domain/3365?camid=4v1a](www.igi-global.com/article/measurement-ontology-generalizable-emerging-domain/3365?camid=4v1a)

A Graph-Based Approach for Semantic Process Model Discovery
[www.igi-global.com/chapter/graph-based-approach-semantic-process/58622?camid=4v1a](www.igi-global.com/chapter/graph-based-approach-semantic-process/58622?camid=4v1a)

INDUSTRY AND PRACTICE: A Challenge to Database Researchers
[www.igi-global.com/article/industry-practice-challenge-database-researchers/51149?camid=4v1a](www.igi-global.com/article/industry-practice-challenge-database-researchers/51149?camid=4v1a)

Evaluating the Performance of Dynamic Database Applications
Zhen He and Jerome Darmont (2006). *Advanced Topics in Database Research, Volume 5* (pp. 294-319).
[www.igi-global.com/chapter/evaluating-performance-dynamic-database-applications/4398?camid=4v1a](www.igi-global.com/chapter/evaluating-performance-dynamic-database-applications/4398?camid=4v1a)