

# Standard Metamodel for Object Databases: Roles and Issues

(Extended abstract)

Piotr Habela\* and Kazimierz Subieta\*<sup>+</sup>

<sup>\*)</sup> Polish-Japanese Institute of Information Technology, Warsaw

<sup>+) Institute of Computer Science PAS, Warsaw</sup>

piotr\_habela@pro.onet.pl, subieta@ipipan.waw.pl

In this paper we investigate the required features of a standard object database metamodel. We present a role of a metamodel in an object-oriented database management system and compare it with the proposal defined in the ODMG standard. After outlining the metamodel definition included in the standard we briefly identify its main drawbacks. We suggest several changes to the ODMG metamodel definition, which we consider necessary to make this part of the standard easier to understand, practically implementable and fully functional.

## 1. The roles of a metamodel

A metamodel definition presents an important part of the construction of a database management system or a corresponding standardization document. Thus it requires careful treatment, especially in case of object databases, whose rich data model inevitably increases the metamodel complexity [3]. The following three major roles of metamodel have to be addressed:

- **Description of a data model.** It provides better understanding of the data model. The metamodel specifies interdependencies among concepts used to build the model, some inherent constraints, and abstract syntax of corresponding data description statements; thus it supports intended usage of the model.
- **Implementation of DBMS.** The metamodel determines organization of a metabase (usually called in this context a database catalog or a data dictionary). It has to be internally implemented in DBMS as a basis for internal database operations, including database administration.
- **Generic programming.** The metamodel plus appropriate access functions become a part of the programmer's interface to enable generic programming through reflection.

These roles of a metamodel are to some extent contradictory. The following sections present requirements connected with each of the mentioned roles.

### 1.1. Description of data model

From among of mentioned roles, the descriptive function of metamodel is probably the most straightforward and intuitive. It is important to provide possibly clear and unambiguous definition of data model primitives and their interrelations. As an example we can mention the Unified Modeling Language (UML) [1], whose metamodel provides quite useful and expressive (although informal) definition of the meaning of language constructs. Formally, this style of definition suffers from the *ignotum per ignotum* logical flaw (i.e. concepts are defined through undefined concepts; definitions are interdependent, but not recursive). Proper use of natural language tokens (commonly understood) reduces this drawback. It is obvious that such metamodel is unable to describe formal data semantics (i.e. valid database states); instead, it presents abstract syntax of data description statements and various dependencies and constraints among introduced concepts.

The ODMG standard [2] follows to some extent this style. It presents interdependencies among concepts introduced in the object model and covers the abstract syntax of ODL. This role of the metamodel is not officially supported. The definitions (presented as interfaces in ODL) are under-specified, not supported by graphical diagrams, without definitions of introduced terms, and with no example.

## 1.2. Metamodel in DBMS implementation

Considering the second role of a metamodel, we distinguish the following criteria to evaluate their quality:

- **Simplicity.** The metamodel should be simple and easy to understand to be efficiently used by developers of a DBMS and by database administrators. Too sophisticated metamodel would be error-prone and its maintenance could be difficult.
- **Universality.** Metamodel should support all kinds of operations allowed to perform on the database.
- **Performance issues.** Calls to the metamodel coming from database management system or from applications could be very frequent, so it is important to organize it in a way that may guarantee a quick run-time access.
- **Physical data structure information support.** Data describing physical structure (e.g. file organizations, indices, etc.) as well as data used for optimization have to be included in the metamodel repository.

The description of the metamodel in the ODMG standard follows this role. According to the official statements, it should have the same role as the Interface Repository of the CORBA standard [4], which presents some data structures together with operations (collected in interfaces) to interrogate and manipulate this structure. The primary goal of the Interface Repository of CORBA is dynamic invocations, i.e. generic programming through reflection. It is not clear if the same goal is supported by ODMG, because the standard does not defines all necessary properties which could make such generic programming possible (see the the next subsection).

ODMG fails to address this role of the metamodel. The definition provided in this document consists of 31 interfaces, 22 bi-directional associations, 29 inheritance relationships and 64 operations, which make it itself a very complex database. The definition is redundant and ambiguous - it would be difficult to understand and use by programmers. The future extensions of this metamodel (e.g. triggers, views or DB procedures, which currently are not properties of the ODMG standard) would be problematic. The standard does not mention other necessary applications of the metamodel, such as specification of physical database structures and query optimization information. The metamodel fixes some particular methods to access and update meta-repository. This collection of methods is not described at all; the reader has to induce the meaning from names and parameters used in the specification. There are obvious examples that this collection of method is not able to fulfill all possible requests.

## 1.3. Support for reflection

To allow generic programming of applications through reflection, the following four steps are to be supported:

1. Accessing the meta-information to retrieve data necessary to formulate a request. As previously mentioned, this require access to metamodel to determine all necessary elements of the data structure description.
2. Dynamic construction of the request, e.g. as a string representing an OQL statement.

3. Executing the request. This assumes invocation of special utility, taking the request as a parameter. The result is then placed in specially prepared data structure. Since a request is usually executed several times, it is desired to provide a preparation function, that stores the optimized request in a convenient run-time format.
4. Utilizing the result. In more complicated cases the type of the result is unknown and has to be determined by a special utility which parses request against the metamodel information.

Although the ODMG standard specifies the means to access meta-information, thus supporting the step 1, it does not provide any support for the subsequent steps (for detailed discussion see [5]).

Especially interesting requirements are connected with step 4. In order to guarantee that in any case the result type can be determined, the returned data have to be connected with the structure describing its type. Such features are available for example in both SQL-89 and SQL-92 standards through the “describe” statement in the dynamic SQL. The ODMG standard does not specify that aspect of metamodel, making an important subset of generic programming tasks unimplementable.

Of course, to allow generic programming and portability, standard has to be very precise in every of mentioned aspects, including uniform treatment of steps 2 and 3. Even subtle differences in the organization of database dictionaries, their access operations, or request execution functionality would make the portability of generic applications impossible.

## 2. Postulated Improvements

As can be seen, the main problems with the current metamodel definition results from its size and redundancy, making it too complicated for implementation and usage by programmers. Therefore our suggestion is to reduce the number of constructs, to avoid the “metadata nightmare” mentioned by Won Kim [3].

There are several options. The most obvious improvement in this direction is removal of concepts that are redundant or of limited use. As an example, we can postulate to remove the *set* concept, because the *multi-set (bag)* covers it and the application of *set* is marginal (SQL does not deal with *sets* but with *bags*). Another recommendations that can considerably simplify the metamodel (as well as a query language) concerns *object relativism*. It assumes uniform treatment of every data element. Thus, differentiating between the concepts of object, attribute, subattribute, becomes secondary. Some simplifications can also be expected from the clean definitions of the concepts *interface*, *type* and *class* concepts and their interrelations.

The next step toward simplifying the metamodel definition concerns “flattening” of its structure. A similar solution has been introduced in the UML metamodel, where the size of metamodel was significantly reduced due to using meta-attributes and the extension mechanism of stereotypes. For example in the UML an aggregation, composition and a regular association are not separate constructs; all are defined by the metaclass *Association* and their kinds are determined by the value of attribute contained in metaclass *AssociationEnd*. This illustrates the concept of moving some part of meta-metadata layer into the metadata layer. Any relationships between repository elements could be expressed as string values. For example, separate metamodel constructs like *Parameter*, *Interface* or *Attribute* can be replaced with one construct, say *Metaobject*, equipped with additional meta-attribute *kind*, whose values can be strings “parameter”, “interface”, “attribute”, or others, possibly defined in the future. This solution would be very advantageous both for performance of metamodel requests, for extendibility, and for maintenance of the metadata repository.

To simplify the functionality offered by the metamodel and to allow its further extensions, a standard, generic set of operations for metadata search and manipulation should be defined. We consider the predefined set of methods as a bad solution. Instead, the interface could be based on generic operations, for instance an OQL-like query language, extended with facilities specific to the metadata queries. It should be possible to define new methods (views, procedures, etc.) rather than to rely on a set of predefined ones.

### **3. Conclusions**

The metamodel with a data repository based on it are necessary to implement internal operations of DBMS. Such repository is also a proper place to store physical data structure information and other information needed for optimization.

Another important role of metadata repository is to support generic programming through reflection. Besides the precise definition of facilities for constructing and executing the dynamic request, special attention should be put to the problem of the query result metamodel.

Although the ODMG provides the definition of a metamodel, the solution is incomplete, ambiguous and in many aspects invalid from pragmatic point of view. To make it useful, the metamodel has to be radically simplified, both by reducing redundant concepts and by “flattening” its structure. It seems to be a necessary condition for avoiding metadata access and management nightmare. Such an approach would also simplify possible future extensions to metamodel. It is desirable to define generic access mechanisms to metadata repository, not limiting its functionality to the set of predefined operations.

### **4. References**

- [1] G. Booch, I. Jacobson, J. Rumbaugh. *The Unified Modeling Language User Guide*, Addison-Wesley, 1998.
- [2] Cattel R., and Barry D. (eds.) *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, 2000.
- [3] Kim, W. Observations on the ODMG-93 Proposal for an Object-Oriented Database Language. *SIGMOD Record*, 23(1), 1994, pp.4-9.
- [4] Orfali R., and Harkey D. *Client/Server Programming with Java and CORBA*, Wiley, 1998.
- [5] Roantree, M, and Subieta, K. *Generic Applications for Object-Oriented Databases*. Submitted for publication, November 2000.