

Overcoming the Complexity of Object-Oriented DBMS Metadata Management

Piotr Habela¹, Kazimierz Subieta^{2,1}

¹Polish-Japanese Institute of Information Technology, Warsaw, Poland

²Institute of Computer Science PAS, Warsaw, Poland

habela@pjwstk.edu.pl

subieta@ipipan.waw.pl

Abstract. The lack of a broader acceptance of today's pure ODBMS (as represented by the ODMG standard) brought many questions concerning the necessary changes of their architecture. One of the issues worth reconsidering is the design of a metamodel that would make metadata operations simpler and the underlying data model more evolvable and extensible. In this paper we discuss the implications of proposed simplified generic metadata structure for an object database. The role of a query language in metadata management is emphasized. We also consider the issue of compliance with existing metamodels. We argue that the simplified structure not only improves database metamodel flexibility, but it also contributes to a more intuitive metadata access.

1 Introduction

Although the usage of commercial Object-Oriented Database Management Systems (ODBMS) remains limited to relatively narrow niches and their standardization scope [1] gradually shrinks towards simpler persistence layer of programming language [11], the domain remains an important and promising area of research aimed at improving the means of data management accordingly to current needs. This situation (that is, limited scope and influence of existing standards) encourages the search for solutions less tightly bound with existing specifications. This is the case for our research on object databases, differing significantly from the approach of the ODMG standard, which we treat as a commonly known starting point for discussing alternative solutions. One of important issues of a DBMS construction is the metamodel definition and related means of metadata management. At the same time this is the aspect where we found the solutions of the existing standard significantly inadequate, as described in [2]. One of the issues identified was an overly complex metadata structure, which results in the lack of flexibility and complicates its usage by programmers (e.g. during programming through reflection).

This paper further discusses the pros and cons of the radically simplified metadata structure originally proposed in [2], taking into account the issues like the usage of a query language to handle metadata as well as the influence and requirements of other standards dealing with metadata. The paper is organized as follows. Section 2 summarizes our assumptions concerning the desirable changes to the ODBMS data

model and language support. Section 3 provides an overview of existing object-oriented metamodel specifications relevant to the subject. Section 4 describes the proposed simplified structure and its impact on database metadata management. Section 5 concludes.

2 ODBMS – Changing the Perspective

This section explains the context of our proposal. The assumptions making it different from the approach represented by the ODMG standard [1] are briefly described.

2.1 ODBMSs – the Low-level Solutions?

The development of popular object-oriented programming languages strongly influenced the shape of the object database standard. The idea to provide a support for the direct storage of objects from several programming languages led to the solution inspired by the OMG CORBA (Common Object Request Broker Architecture) [5] middleware standard that defines a common object model and a number of bindings to supported programming languages. On the other hand, to provide possibly high-level means of data access, the Object Query Language (OQL) has been defined and the pattern of embedding queries within a programming language code known from SQL was followed. Unfortunately, despite the alignment of language constructs, the issue of so-called impedance mismatch of such programming language – query language join was not completely removed. Additionally, because of the complexity of OQL¹ its usually only partly implemented or just absent in the majority of commercial systems following the ODMG standard. In such case a general-purpose programming language remains the only mean of accessing stored data. Taking into account the importance of SQL for the success of relational DBMS technology, the lack of analogous high level declarative mean of data access has to be assumed as a serious drawback of object database systems.

For those reasons we assume the need of introducing an object query language extended to provide full algorithmic power and imperative constructs (similarly like Oracle's PL/SQL extends the standard SQL). This would reverse the existing pattern in the sense that regular programming languages would become secondary means of data manipulation. The assumption is important for our discussion since it would make such query language a natural choice also for metadata retrieval and updating.

2.2 Redundancy of Data Model Constructs

It is intuitive that the expressiveness of an object data model requires supporting a number of notions, which results in inherent complexity of object metamodels,

¹ A promising approach to this issue is the Stack-Based Approach [9]. Its ability to create clear operational definitions of query language operators allowed us to successfully implement several prototypes, including those supporting newer notions like dynamic object roles. [3]

compared with e.g. the relational model. This can be assumed to be a natural cost of the more powerful mean of modeling the information. However, the difficulties of metadata management within a database schema strongly motivate steps towards minimizing the number of concepts.

From this point of view, the idea to unify the data model notions of all supported programming languages within an ODBMS data model (as assumed by the ODMG) does not seem to be optimal. The data models of particular programming languages differ, which makes the clean definition of a common object model problematic. This is especially visible for some C++-specific notions (*structs*, *unions*), which make the resulting data model of rather hybrid nature.

The lack of minimality of such data model additionally motivates the thesis that if an ODBMS has to be universal concerning its interoperability with general purpose programming languages, it should rather define its own, possibly clean data model instead of directly supporting those languages' features.

3 Object-Oriented Metamodels

This section provides an overview of the most popular proposals concerning object-oriented metamodels that are relevant to our discussion.

3.1 The UML Specification Family

The Unified Modeling Language (UML) provides a graphical notation for visualizing, specifying, constructing, and documenting the artifacts created at different phases of a software development process [8]. The language defines a rich set of modeling notions together with graphical notation elements used to visualize them. The part of the standard most interesting for us, and at the same time the central element of the whole specification are class diagrams, as they define such data-definition language-related notions like class, attribute, association, operation etc. Although UML is programming language-neutral, its object model is especially influenced by C++ and Java languages. Because of the success of this specification, the main object-oriented notions are very often understood in the terms assumed by the UML's object model.

UML is also a source of probably the most popular example of object metamodel. This metamodel is defined in the so-called *metacircular* style, that is, the concepts of the language are defined in terms of the UML itself. The notions of the language are defined as [meta]classes. The attributes, associations and generalization relationship are used to describe the concepts and their interdependencies. Remarkably, the definition is fully static in the sense that the metaclasses forming the specification do not use operation declarations. To specify additional constraints not expressible by standard graphical notation elements, a precise declarative (and state-preserving) language named OCL (Object Constraint Language) has been introduced.

The UML Object Model (being technically a subset of the core elements of the language) provides common object-oriented notions for a family of specifications. Most remarkably, it served as a pattern for the MOF (Meta Object Facility) specification, being the framework for managing technology neutral metamodels [7].

This specification may play an important role in supporting an effective exchange of data and metadata among different systems. As a central element of the recent MDA (Model Driven Architecture [6]) initiative, it is also expected to form a base for a high-level approach to software development.

Taking into account the importance of the UML-related standards and the popularity of their definition of the object data model concepts, it is highly desirable for an ODBMS solution to keep a level of compliance with this object model. On the other hand, due to its different purpose, the UML metamodel cannot be compared with DBMS metamodels. Particularly, it does not define an object store model needed to precisely describe, how metadata and its instances are represented within a DBMS.

3.2 OMG CORBA and Interface Repository

CORBA (Common Object Request Broker Architecture) [5], defined by the OMG (Object Management Group) consortium is an object-oriented middleware standard, allowing programmers to abstract from several aspects of a distributed software interaction. The standard defines common language-neutral concepts² used to define software interfaces and supports a number of different programming languages, by defining their mappings of those concepts.

The standard defines means of accessing objects' metadata through the facility called *Interface Repository*. This allows to dynamically determine the type of a remote object and extract its interface definition; that is, to collect metadata necessary to construct a dynamic call to its properties.

Each interface definition has assigned its repository identifier, which allows to maintain the identity of such metadata in presence of multiple repositories. Version number of an interface is also stored, although the definition versioning is not supported by any additional mechanism nor semantics [5]. A particular interface definition can be located in three ways:

- Directly from the ORB (e.g. through the invocation of *get_interface()* operation on CORBA Object);
- By navigation through the module name spaces (that is, by interface name);
- By lookup of a specific identifier (that is, by an ID, which may be useful to find a definition corresponding to another) [5].

With presence of full metadata manipulation functionality, the consistency of the repository presents a hard problem. Indeed, only the most obvious inconsistencies (like e.g. name conflict within one interface definition) can be immediately detected and reported. Thus, the means of direct updating of the metadata are provided at the cost of leaving the consistency of a repository practically unprotected.

Including recent extensions towards the component model, the Interface Repository specification now consists of nearly 50 interfaces and a number of structure types,³ which constitutes a really complex structure to be queried and manipulated by programmers. Moreover, it is assumed that further extensions (both

² However, the influence of the C++ language on the design of the specification is predominant.

³ The use of a structure (*struct*) as an operation's result instead of object reference is motivated by the cost of remote call: a structure provides a chunk of data that can be processed locally.

defined by future standard's versions as well as custom, domain- or tool-specific extensions of standard defined interfaces), would be introduced through the specialization of existing definitions.⁴

Despite significant complexity, the community seems to accept the solution, as being a natural consequence of overall standard's assumption, to provide a possibly direct support for a number of existing mainstream general-purpose programming languages. However, the programming against the Interface Repository is commonly perceived being rather complicated or at least inconvenient.

3.3 The ODMG Metamodel

The ODMG metamodel [1] is defined through a collection of ODL (Object Definition Language) interfaces, which are intended to provide access to an ODBMS schema repository, organized analogously to CORBA's Interface Repository (IR), which has been the pattern for this definition. The structure of the metamodel is very complex, and despite the total number of interfaces (31) is smaller compared to the latest version of Interface Repository specification [5], its usage seems to be even more complicated. The style of metamodel definition with its extensive use of associations and generalizations to some extent resembles the UML specification, however in contrast to the latter it defines a large number of narrowly specialized operations (including updating ones), which contributes to the complexity of this metamodel.

Considering the characteristics of a database schema like e.g. the need of frequent (and not necessarily remote) querying or the requirement of extensibility to store additional metadata, we find the idea of directly adapting the CORBA Interface Repository the ODMG followed, to be an inadequate solution.

The ODMG metamodel specification still seems to be immature, and the consequences of providing some of its features (e.g. metadata-updating operations) have not been addressed so far. This is confirmed by the fact that e.g. the latest ODMG Java binding specification has not accommodated those interfaces at all, while the C++ binding supports them only in their read-only part.

While the ODMG standard metamodel remains the most relevant specification to the topic of this paper, the mentioned inadequacies suggest considering significant changes to its approach.

3.4 Web-related Solutions and the Requirements of Extensibility

Although the Web-related solutions like XML (eXtensible Markup Language) [14] or RDF (Resource Description Framework) [13] are not directly relevant to the subject, the recent development of database technology brings the challenges that make the main principles of Web technologies worth following.

The mentioned trends concern especially a bigger interest in data sources' interoperability in distributed and heterogeneous environments and growing

⁴ As stated in the standard specification, the IR is intended to store additional interface-related information like e.g. debugging information, libraries of related connectivity code etc. [5].

importance of effective standardization. This requires from database technologies specifications support for various extensions and the ability of future evolution.

Those requirements make the design principles of Web technologies (as stated by the World Wide Web Consortium (W3C) [12]) fully relevant also to the database domain. The following principles are emphasized [12]:

- **Interoperability.** The ability to cooperate with the broadest selection of other solutions in the field. In case of database systems this would mean an effective handling of different kinds of heterogeneity among particular DBMSs as well as the differences of database designs.
- **Evolution.** The solution should be prepared to accommodate future technologies. To make the future changes feasible, the specification should follow the qualities of simplicity, modularity, and extensibility.
- **Decentralization.** The lack of dependencies on central resources is highly desirable. The development of distributed and federated database systems to some extent follows this trend.

The success of Web-related technologies based on lightweight and extensible structures encourages us to follow a similar approach in reconsidering the design of the ODBMS metamodel.

4 Simplified Metadata Structure

The doubts concerning the complexity of ODBMS metadata management have been raised from the beginning of their history, when the remarkable term “metadata management nightmare” appeared [4]. In fact, as shown in the previous section, applying the object modeling principles to meta-modeling of an ODBMS itself results in a quite complex metabase.

We find it problematic concerning the ability of future metamodel evolution, its extensibility (e.g. to store custom, not standardized metadata) and convenience of the access to a schema repository. To solve this problem we have proposed the radically simplified metadata structure [2], realizing the postulates of genericity and extensibility in the spirit similar as in case of some Web technologies (XML, RDF) [13,14]. In the following sub-sections we describe the solution and motivate our choice, showing its ability to fulfill typical requirements towards metadata structure.

4.1 The Flat Metadata Structure

The proposed solution (see [2] for details) achieves the abovementioned goals through the flattening of original metalevel hierarchy in the sense that meta-metadata (e.g. the term “Interface”) and metadata (e.g. the term “Person”) coexist at the same level of implemented schema structure. As a result, 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 like e.g. “parameter”, “interface”, “attribute”, or others, possibly defined in the future.

This approach radically reduces the number of concepts that the metadata repository must deal with. Moreover, it supports extensibility, because a new concept means only a new value of the attribute “kind”.⁵ The metabase design could be limited to only a few constructs, as demonstrated in Fig. 1. An example of meta-attribute could be “*isAbstract*” attribute of metaobject describing *Class*. The *kinds* of meta-relationships would probably include “*generalization*” and “*specialization*” to connect metaobjects of the *Class* kind. Although this meta-schema does not support some useful concepts (e.g. complex meta-attributes, attributes of meta-relationships), it constitutes a sufficient base for straightforward definition of the majority of constructs required to express ODBMS metadata.

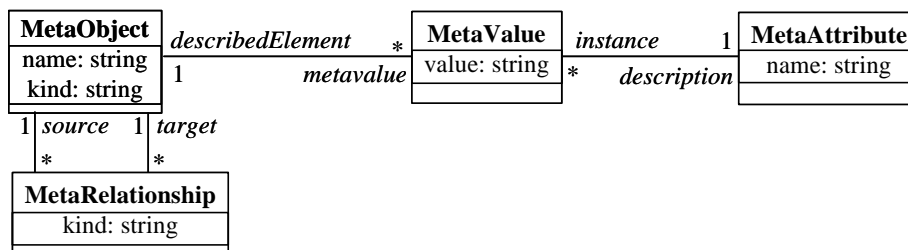


Fig. 1. Concepts of the flattened metamodel

It should not be a surprise that any example of metadata defined according to this structure (that could be visualized e.g. through UML’s object diagrams) results in a larger graph than it could be in case of traditional, rich metamodel structure. This is a rather obvious cost of the achieved genericity and reduction of constructs.

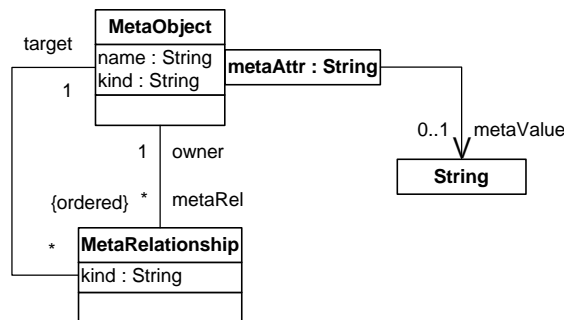


Fig. 2. Compact implementation structure of the metamodel. The qualified association is meant to express an associative array, mapping meta-attribute names into their values expressed as strings.

However, since the structure proved to be quite intuitive, we accept this overhead and make only minor adjustments in order to make the metadata graph more concise.

⁵ Of course any newly introduced notion would require implementing a proper support from ODBMS mechanisms. Anyway, from the point of view of the metamodel maintenance the structure offers maximum flexibility.

Namely, the experiences with mapping existing metamodels into the flattened form suggest the following changes (see Fig. 2):

- Since the application of repeating (multi-valued) attributes in meta-modeling is relatively small, we could give up supporting them. This allows for much more convenient handling of meta-attributes through the use of associative array of strings, indexed with meta-attribute names.
- Although the ordering of links towards meta-relationships owned by particular metaobject is often not necessary, in some cases it can be very helpful. For example, this would allow to avoid introducing a meta-attribute named *position* into a metaobject describing method's parameter.

4.2 Constraining the Generic Structure

The large part of the presented metadata is used to distinguish appropriate object data model constructs. In order to define a standard metamodel, the flattened metamodel has to be accompanied with additional specifications, which should include:

- Predefined values of the meta-attribute “kind” in the metaclass “MetaObject” (e.g. “class”, “attribute”, etc.); they should be collected in an extensible dictionary.
- Predefined values of meta-attribute names (e.g. “isAbstract”) and of meta-attributes “kind” of “MetaRelationship” metaclasses (e.g. “specialization”) – for the latter the name of reverse meta-relationship can be specified if applicable.
- Constraints defining the allowed combinations and context of those predefined elements.

When experimenting with a generic metadata repository, we have developed the following “dictionary” in order to define particular metamodels (see Fig. 3). Although a DBMS does not need this level of genericity (as its implementation assumes particular metamodel, to large extent hardwired into it), this structure makes it explicit, what constraints need to be controlled in case of the flat metadata structure.

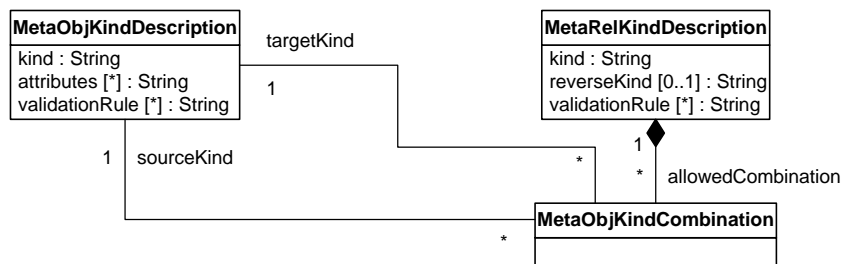


Fig. 3. The “dictionary” structure for defining the notions of particular metamodel over the generic flat metadata structure

As can be seen, the definitions presented above form together a metamodel framework of features analogous to those of traditional metamodels (like ODMG or UML). In other words, despite the specific form of storage, the conceptual distinction of metalevels remains unchanged. Thus, the mapping into other standard metamodels can be straightforward. This is desirable for the following reasons:

- Before formulating the final form of his/her model, the developer should be able to take advantage of existing modeling tools based on the UML, as they provide a more expressive form, optimum for conceptual modeling.
- It may be important to be able to easily map the notions of an ODBMS metamodel into a common description (e.g. in terms of the OMG MOF), in order to support interoperability of data sources using different data models.

M 3 : Standard common meta-metamodel (e.g. OMG MOF)

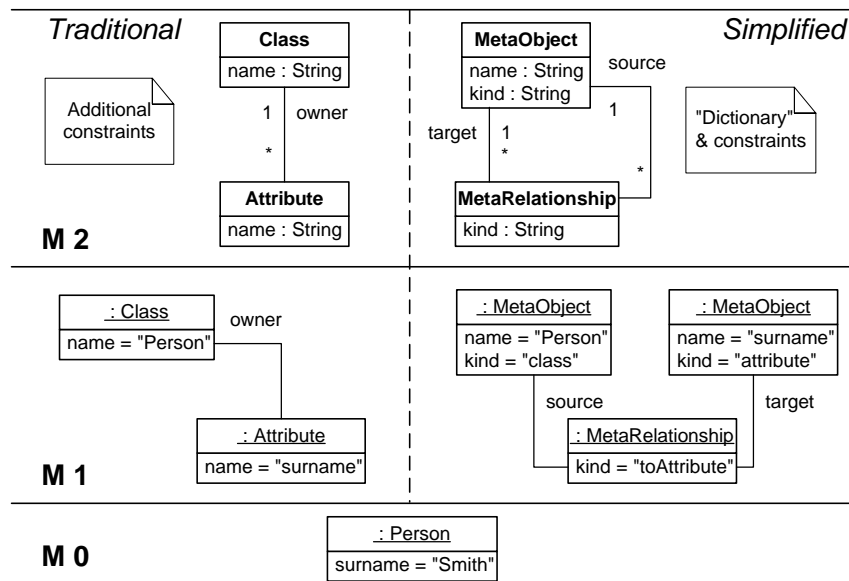


Fig. 4. The simplified metamodel illustration within the 4-level metamodel architecture

Fig. 4 presents an example fragment of the simplified metadata structure in the context of the 4-level metamodel architecture (assumed e.g. in OMG standards [7,8]). As can be seen, the simplified metamodel has the following properties:

- It has no direct impact on the M0 (regular objects) level. Particularly, no limitation is imposed on the data model notions used to represent regular objects.
- Similarly, the flattening of metamodel does not affect the meta-metamodel (M3), which can be used to describe the simplified metamodel, other metamodels and mappings among them.
- The flat metamodel itself (M2) needs to be to larger extent supported by constraints specific for particular metaobject kinds (described through the means similar to the structure from Fig. 3).

4.3 The Limitations of the Simplified Structure

Despite the radical simplification of its structure, the metamodel remains conceptually complex and its well-formedness needs to be protected with additional constraints.

The most obvious limitation of the flat metamodel is the inability to use metadata kind-specific operations (e.g. *giveSuperclass()* operation of *Class*), since all metaobjects are instances of the same class. We intentionally give up such approach in favor of generic means of metadata access, discussed in the next subsection. The absence of specialized classes also means that metaobjects are not partitioned accordingly to their kind, although other options (e.g. indices) can be used to ease their lookup. Finally, the flattened structure cannot take advantage of generalization relationship. This means that for example in a query about *operations* and *attributes* of a given class, a single name (say, *property*) could not be used.

4.4 Accessing Metadata

As mentioned, the flat metamodel (in contrast to the ODMG specification) does not resort to metaobject kind-specific methods. We assume using generic operations of a query language instead. This would make programming through reflection scenario more similar to the one of Dynamic SQL rather than that of OMG CORBA Interface Repository, the ODMG was inspired with. Although the overview is only superficial (as we do not describe here neither a complete metamodel nor query language proposal), below we present examples of possible queries that could be used for a generic programming against the flat metadata structure.

The examples are formulated in the language SBQL (Stack-Based Query Language), which was implemented e.g. in the prototype ODBMS LOQIS [10]. Its advantage over the OQL ensues among others from the following features:

- Precisely defined semantics and a good ability of adapting new operators.
- Concise syntax, also thanks to the lack of the SQL-like syntactic sugar.
- Full algorithmic power and the presence of imperative constructs.
- Availability of recursive (transitive closure) operators.

Analogously as in case of the CORBA Interface Repository, we may assume three ways of accessing metadata: direct querying, access using a retrieved reference and reflecting upon a particular object. In the first case the query would usually include the selection of metaobjects of particular kind. For example, to list the names of all registered classes we would need to write the following code:

```
(MetaObject where kind="class").name
```

Similarly, we may query for metaobjects' properties described by meta-attributes. The syntax in SBQL would depend on the way the meta-attribute dictionary suggested in Fig. 2 is realized. Thanks to the flexibility of the language (ability to deal with arbitrarily nested complex objects and treating object name as the first category item), the representation can be very compact. For example, the meta-attributes can be stored as simple text-valued, named sub-objects of *metaAttributes* complex object, which in turn is contained in particular metaobject. Then, the query returning the number of instances of class "Person" would have the following form:

```
(MetaObject where (kind="class" and name="Person")).  
metaAttributes.numberOfInstances
```

Assuming that every database object would implement a reflective operation *reflect()* returning its class definition, the following code illustrates the possible syntax of the query returning the list of methods (with their names and return types), which are directly defined by the class of the processed object:

```
(myObj.reflect().metaReIs.MetaRelationship
where kind="toMethod").target.MetaObject.
name, (metaReIs.MetaRelationship where
kind="toType").target.MetaObject.name)
```

In this example the class of the object stored in *myObj* variable is extracted. Then a collection of references to its meta-relationships pointing at methods' definition is selected. The references are explored and metaobjects describing methods are accessed. There is no need of checking the kind of those metaobjects, since we assume that metaobjects of kind "method" are the only legal targets of the meta-relationships of kind "toMethod". Analogously the further navigation leads to discover those methods' names and return types.

Another example shows the importance of the transitive closure operator (named *close by*) for metadata querying. Assuming that the *reflect()* method would return object's most specific class, the navigation up the generalization hierarchy would be necessary to list the names of all attributes the object possesses:

```
(myObj.reflect() close by ((metaReIs.MetaRelationship
where kind="generalization").target.MetaObject).
metaReIs.MetaRelationship where kind="toAttribute").
target.MetaObject.name
```

The evaluation starts from the reference to *myObj*'s most specific class (returned by the *reflect()* operation) and collects recursively the references to all its superclasses. Further navigation proceeds like described above to retrieve names of all attributes defined within those classes (that is, the names of all attributes the *myObj* object has).

As can be seen, despite the lack of specialized methods and even classes for particular kinds of metadata, which makes the paths longer, the access pattern remains reasonably intuitive. Another use of a query language would be the definition of the well-formedness rules for particular kinds of metadata in case they constitute a custom extension and thus require an explicit constraint definition.

5 Conclusions

The initial remark of this paper is the indispensability of changes to the ODBMS architecture assumed by the ODMG standard. The bigger productivity and applicability of ODBMSs seems to require a powerful query language as the central feature of the system. In this context we have proposed the generic flattened metadata structure, which in our opinion is an optimum solution for implementing an ODBMS metamodel, concerning both its extensibility and usability in metadata access.

The shape of the proposed solution raises the questions about its compatibility in the sense of mapping into a modeling language metadata and interoperability with

other metamodels, as well as about the intuitiveness of manipulating such structures. However, as it has been stated, the difference is only of technical nature, that is, it does not affect the conceptual shape of metalevels. Although the flattened structure can be perceived being less expressive (due to minimal set of constructs used), as we attempted to demonstrate, it can be quite effectively accessed using a query language.

In our future research we plan to develop a more complete ODBMS prototype, including schema management based on the proposed structure. This would of course require a precise solution of a number of issues not discussed in this paper. For example, since the SBQL language includes generic update operators, the update statements concerning metadata need to be constrained and specially handled in order to protect consistency of metadata, regular data and application code dependent on it.

Another interesting subject emerges from the current needs of extending databases' interoperability. This requires augmenting the traditional contents of database schema with descriptive statements, converging to some extent with the metadata features assumed by the Web community. Since it is difficult to provide an exhaustive list of descriptive statements that may be needed, the related W3C specification (RDF [13]) offers an open framework based on a very simple structure instead. In this context, the proposed flat metadata structure offers analogous features concerning flexibility and extensibility, which we consider necessary to make an ODBMS specification evolvable both in terms of its core data model as well as various metadata extensions.

References

1. R. Cattell, D. Barry: The Object Data Standard: ODMG 3.0. Morgan Kaufmann 2000.
2. P. Habela, M. Roantree, K. Subieta: Flattening the Metamodel for Object Databases. ADBIS 2002: 263-276
3. A. Jodlowski, P. Habela, J. Plodzien, K. Subieta: Objects and Roles in the Stack-Based Approach. DEXA 2002: 514-523
4. W. Kim: Observations on the ODMG-93 Proposal for an Object-Oriented Database Language. ACM SIGMOD Record, 23(1), 1994, 4-9
5. Object Management Group: The Common Object Request Broker: Architecture and Specification. Version 3.0, July 2002 [<http://www.omg.org>].
6. Object Management Group: Model Driven Architecture (MDA). July 2001 (draft) [<http://www.omg.org>].
7. Object Management Group: Meta Object Facility (MOF) Specification. Version 1.4, April 2002 [<http://www.omg.org>].
8. Object Management Group: Unified Modeling Language (UML) Specification. Version 1.4, September 2001 [<http://www.omg.org>].
9. K. Subieta, C. Beerl, F. Matthes, J. W. Schmidt: A Stack-Based Approach to Query Languages. East/West Database Workshop 1994: 159-180
10. K. Subieta, M. Missala, K. Anacki: The LOQIS System, Description and Programmer Manual. Institute of Computer Science Polish Academy of Sciences Report 695, 1990.
11. Sun Microsystems: Java Data Object (JDO) Specification. Version 1.0. 2002.
12. The World Wide Web Consortium website [<http://www.w3.org/>].
13. The World Wide Web Consortium: Resource Description Framework (RDF) Model and Syntax Specification. February 1999 [<http://www.w3.org/>].
14. The World Wide Web Consortium: Extensible Markup Language (XML) 1.0. October 2000 [<http://www.w3.org/>].