Association-Oriented Database Model — n -ary Associations

Marek Krótkiewicz

*Faculty of Electrical Engineering
Automatic Control and Informatics
Opole University of Technology
ul. Prószkowska 76, 45-758 Opole, Poland
m.krotkiewicz@po.opole.pl; mkrotki@mkrotki.com*

Received 1 September 2014

Revised 8 October 2014

Accepted 5 March 2016

This paper provides a comparison between methods used to represent unary, binary and n -ary relations in the relational database (RDB) and object-oriented database (OODB) models and the association-oriented database (AODB) model proposed in terms of their unambiguity. We have elaborated upon the RDB, OODB and AODB models perceived from the perspective of the categories applied to define them. What has been discussed is a number of solutions assuming the form of schemes for the AODB model. We substantiate that those solutions have unequivocally met the conditions of relations of any chosen arity and multiplicity, at the same time keeping the notation concise, unambiguous and considerably expressive.

Keywords: Association-oriented; database model; n -ary associations.

1. Introduction

Database applications are by far the largest of software groups developed. From the perspective of software engineering and diverse methodologies of the computer program development, the very core is the component responsible for data storage. Databases understood as models have been developed for many years [25]. They evolved starting from a hierarchical model, through a network model to finally become the relational database (RDB) model that has prevailed for years now [2]. The RDB model is based on mathematical foundations and it is also extremely simple in terms of the number of categories used. Some theoreticians consider it to be particularly elegant or even optimum. One could recently observe predominance of the

This is an Open Access article published by World Scientific Publishing Company. It is distributed under the terms of the Creative Commons Attribution 4.0 (CC-BY) License. Further distribution of this work is permitted, provided the original work is properly cited.

object-oriented paradigm in the field of programming. As the significance of this approach was increasing, also the problem referred to as impedance mismatch was growing in importance in the design and implementation of database applications. Its main assumption is that applications are usually written using object-oriented languages, whereas the data definition and manipulation is handled by means of SQL. SQL is a declarative language having literally nothing in common with the object-oriented approach, and this fact is to a large extent responsible for the dissonance in program codes [6, 7, 29]. In the 1990s, an idea came into existence to develop a database model compatible with the programming model, i.e. the object-oriented one. The concept was and still is extremely tempting owing to the model cohesion across the entire application [3, 4].

Unfortunately, the agenda of adopting a single obligatory standard for the object-oriented database (OODB) model has been unsuccessful so far, which may certainly be attributed to a variety of reasons [27, 28]. In 1999, a document was developed, only to be published in 2000 as ODMG 3.0 version [5], that was to become the very foundation of the object-oriented database standard. It mainly contained some fundamental principles aimed to reconcile different approaches. It was also a piece of work full of compromises. The standard described in ODMG 3.0 is difficult to implement, whereas in terms of the *Object Query Language* (OQL) — virtually unimplementable [30]. The standard in question, or rather an attempt to establish one, was a failure, and the document itself became a mere reference paper. Both theoreticians and practitioners tend to refer to it while demonstrating differences compared to the solutions they propose.

One of the most fundamental and crucial database-related problems is the aspect of relationships in terms of their arity and multiplicity. The major goal of the paper is to present the importance of n -ary relations in the modeling of the problem that has its representation in database schemas [17, 22–24]. In this paper, we have discussed selected scheme ambiguity aspects of relational and object-oriented databases. Moreover, association-oriented database (AODB) model has been proposed, as new and original database model. Solutions provided as database schemas present properties of the new model, especially in the aspect of n -ary relationships unequivocal definition. There has also been an aspect of relationship multiplicities presented, that provides higher expressive power then compared to other database models.

AODB is a model developed by us, stemming from the general idea of *entity-relationship* (E-R) model [1] as well as an ODMG *object model* (OM) [5]. The proposed model has been verified through direct implementation in C++ language, therefore it is fully integrated with it. There have also been prepared solutions for other OO languages, e.g. *Python* binding.

AODB is a new database model, therefore Sec. 2 provides a short motivation that led us to the development of this model, as well as information regarding the basic concepts and solutions of AODB. Section 3 presents the comparison of the AODB and E-R model. Section 4 is a thorough study of unary, binary and n -ary relations constructs in all three compared models, while Sec. 5 presents summarization of that

study. Section 6 consists of the discussion provided by us regarding the use of different relationships constructs. Section 7 concludes the paper.

2. Motivation

The original motivation to develop an AODB model was a necessity to build a database layer conforming to the needs of knowledge processing. The research project referred to as a Semantic Knowledge Base (SKB) assumes that the basic knowledge category is a concept understood as a set of relationships to other concepts [8]. SKB is a multifaceted knowledge representation and processing project featuring numerous dedicated modules: *ontological module*, *extended semantic networks module*, *time and space module* and *the linguistic module* [9]. The degree of its complexity as far as the data structures applied are concerned is very high. For design and efficiency purposes, it was necessary to ensure a data storage layer meeting the SKB requirements, especially in the aspect of unequivocal definition of n -ary relationships along with various multiplicities of them. Attempts were made to use relational databases and selected implementations of object-oriented databases, however, they were little successful, and the main issue encountered was a relatively low power of expression of these models as well as yet another significant issue of scheme ambiguity. Based on the research conducted and the experience acquired, several basic assumptions for a new model — **association-oriented database** — have been developed:

- Idea
 - Abandoning quantitative minimalism in terms of categories;
 - bringing conceptual, logical and physical modeling down to one level;
 - the notion of association in the very center of the association-oriented modeling idea;
 - principle: one model element should perform one function in the model;
 - principle: one operation should refer to one model element; and
 - abandoning fictitious categories occurring in the conceptual sphere only, e.g. aggregation in object-oriented modeling.
- Structure
 - Categories: collection (*Coll*), attribute (*Attr*), object (*Obj*), association (*Assoc*), role (*Role*) and association-object (*AssocObj*);
 - n -ary relationships established on a physical level;
 - comprehensive approach to multiplicity of relationships (multiplicity also in terms of an association); and
 - descriptions of associations and roles.
- Mechanism
 - Independent role and attribute inheritance;

- new inheritance mode;
- new, unambiguous algorithm for defining prefixes of attributes and roles, ensuring full access to all components inherited; and
- redefining the component virtuality mechanism.

This paper is but a detailed discussion of one of AODB aspects, namely the ability to establish n -ary relationships on physical level. This idea stems directly from E–R model approach, where two categories were defined on extensional level, i.e. entity and relationships between them, while on the intentional level E–R has defined entity sets and relationships between these sets, respectively. At this point E–R has no direct implementations, therefore we in the following section compare AODB to those models that not only have been conceptually defined but have also been implemented and are being used nowadays by developers. Nonetheless, we provide concise comparison of E–R and AODB metamodels in order to present their similarities and differences.

Appendix A is a compact reference to categories defined under AODB and provides insights of the model. For the sake of introduction and clarification, several most crucial aspects of the AODB model should at least be mentioned in order to avoid further terminological misunderstandings. Roles ensure possibilities for establishing relationships between an association and a collection. An association may feature n roles, where $n > 0$. Hence the conclusion that there are unary, binary, ternary, etc., or generally speaking, n -ary relationships, in AODB. Associations are also subject to the inheritance mechanisms, and so roles may be inherited in the same way as attributes are in collections.

Appendix B contains a case study of exemplary database modeled within AODB metamodel. Moreover, there has been presented an AODB Admin Panel application, which provides the tool for graphical database modeling as well as automatic database structure files creation. AODB Admin Panel utilizes function libraries which implement the AODB metamodel.

3. Comparison of E–R and Features Table

AODB and E–R are constructed in a similar way according to how they approach n -ary relationships. The main difference lays in fact that E–R does not reach beyond conceptual layer, meaning that each implementation of database requires to be translated into another metamodel. In case of AODB, each and every element used during modeling phase is directly and unequivocally implemented (have physical representation) and they do not need to be translated or mapped into any other metamodel. In the following part of the paper we focus on the comparison of RDB, OODB (ODMG 3.0 Standard) and AODB as metamodels having the ability to be implemented directly. Nevertheless, a very short comparison of E–R and AODB is presented in Table 1, not mentioning E–R would possibly leave a gap in the thought process.

Table 1. Comparison of E-R and AODB.

Metamodel features	E-R (1976)	AODB (2015)
Separation of data and relationships among them	Yes	Yes
Relationships arity	<i>n</i> -ary	<i>n</i> -ary
Relationships multiplicity	One-to-one, one-to-many, many-to-many	one-to-one, one-to-many, many-to-many
Relationships membership constraint (mandatory, optional)	Yes	Yes
Navigability of relationships	Undefined	One-directional and bidirectional
Inheritance	No	Yes
Virtual inheritance	No	Yes
Selective inheritance of attributes and relationships	No	Yes
Lifetime relationship	Yes	Yes
Modeling language	Yes, many modeling languages	Yes
Query language	No	Yes
Data language	No	Yes
Data storage model	No	Association-oriented, based on hypergraphs native to the model
Implementation	No	Yes
Popularity	Huge	New project

3.1. Relationships navigability in E-R

The entity–relationship metamodel is but a conceptual construct. It has no data storage model defined. Navigability is a structural feature, but it has not been defined in the metamodel since there is no such term in the E–R specification.

3.2. Data storage model in AODB

AODB model specifies the data storage model. It is based on dedicated approach using the concept of hypergraphs. Instances are stored in collections being lists of objects. Relationships are stored in more complex manner allowing efficient access to them. Association-objects are containers for role objects that in turn hold the information on elements fulfilling particular role. In the effect the hypergraph being the basic data structure of AODB is formed. It is not a typical straight hypergraph, since its hyperedges can have many levels of branches. Thus, every hyperedge, in general, can be treated as a tree structure, what allows construction of complex, subtle, intuitive structures that are efficient as well.

4. Conceptual Relationships in Different Models

The considerations provided in this paper are often based on examples clearly depicting the most significant features being discussed. Let us first elaborate upon a

Fig. 1. Sample conceptual unary relationship of a **group**–**member** type in an AODB model.

sample concept referred to as a **group**. One of the interpretations of semantics of this concept is that a **group** is a relationship linking the concept of a **group** with that of a **member**. In conceptual terms, it is a unary relationship where the multiplicity of the **member** is referred to as many, whereas that of the **group** is one. Assuming that the **member** may be related to numerous **groups**, one is left with a question if there can be a **group** without **members**, and whether each **member** must be associated with a **group**. Then assuming that each **member** must be associated with at least one **group**, and that each **group** features at least one **member**, a database scheme of an AODB model can be developed as shown in Fig. 1.

What matters particularly in this approach is that the concept referred to as a **group** has been defined as an association following the assumption that concepts form semantic relations.

Different solutions are also available. In an RDB model, the concepts of a **group** and a **group element** are accordingly represented by two tables, whereas the very fact of their mutual link being established is represented as a limitation imposed upon the values of corresponding *primary keys* and *foreign keys* [20]. It is definitely not intuitive enough and considerably remote from the human perception of reality where no such notions as *primary* or *foreign keys* exist at all.

Figure 2 depicts the database scheme discussed in an RDB model.

The basic category of an OODB model is that of classes. Therefore, both a **group** and a **member** are classes, and their relationship constitutes an association [10, 11].

Figure 3 shows the database scheme discussed in an OODB model.

It does not define the **group** as a concept, i.e. an association in the AODB model. The OODB model features two classes linked by means of an *association*. Regardless of whether the **group** is to be described with a chosen set of attributes or not, in the OODB it is always a class. But it still does not illustrate the most important matter, namely the **group** being an element associating other concepts. In the AODB model, the structures being defined form an association and may be described by a describing collection. However, in order not to depart from the main idea of consideration, this model element has not been discussed further on in this paper.

Fig. 2. Sample conceptual unary relationship of a **group**–**member** type in a relational database model.

Fig. 3. Sample conceptual unary relationship of a *group-member* type in an object-oriented database model.

4.1. Generalized problems of the unary relationship representation

4.1.1. The relational database model

According to the principles of RDB model, physical unary relationships do not exist [12]. The guiding idea of a unary connection can be accomplished as a binary connection by treating one table as an associating table and the second one as a table of elements being associated. Figure 4 shows a sample method of establishing a unary relationship. Table A provides the relationship, and table X contains the related elements. And since table A performs the function of a relationship, it does not need to have any attributes, as such, and it must only feature the *primary key* enabling the very relationship to be established. In the RDB model, the connection is artificially made using a table in which only those attributes that establish the relationship in question are required. Assuming a physical RDB model form, one cannot establish a *many-to-many* relationship, hence this relationship has the multiplicity of one for table A. Adding or removing an element being associated requires adding/removing the corresponding tuple in table X [13]. Depending on the integrity constraints, it is permissible that this tuple is left unconnected or removed in the case of a relationship removal operation. From such a perspective, this operation appears to be rather simple and does not require any reference to several scheme elements, however, it forces one to operate with an attribute or attributes performing a special function, referred to as a *foreign key* in RDB.

For the purposes of the example in Fig. 4:

- Table A constitutes a collection of data performing the function of relationship;
- table X constitutes a collection of data performing the function of related elements;
- R is the relationship functioning as an equality relation: $A.ID_A = X.ID_A$;
- attribute ID_A from table A performs the *primary key* function; and
- attribute ID_A from table X performs the *foreign key* function.

Fig. 4. Sample conceptual unary relationship in an RDB model.

4.1.2. The object-oriented database model

According to the principles of OODB, unary connections do not exist, similarly to the RDB model [5, 14–16]. However, they can be established in a manner analogous to the RDB model.

For the purposes of the example in Fig. 5:

- Objects of class A performs the function of relationship;
- objects of class X are the related elements;
- association P associates objects in class A and class X;
- role R1 determines the associating attribute; and
- role R2 determines the associated attribute.

The association has been presented as two-directional, which is not necessary. In the example considered (see Fig. 5), it has been assumed that the associating element has information on the associated elements, and vice versa. The multiplicity of role R1 is set as one. It has also been assumed that each associated element (object in class X) may only be associated with exactly one associating element (object in class A). There may be numerous associated elements: 0..*.

4.1.3. The association-oriented database model

According to the principles of AODB model, a unary relationship is established in a direct manner. It has but a single role, meaning that objects in the given collection may only perform one role in the given association, which directly relates to the conceptual approach. Figure 6 shows a sample method of establishing a unary relationship in AODB model.

For the purposes of the example in Fig. 6:

- Association A performs the function of relationship;
- collection X contains related elements; and
- role R determines the nature of the association A.

Fig. 5. Sample conceptual unary relationship in an OODB model.

Fig. 6. Sample conceptual unary relationship in an AODB model.

Fig. 7. Sample unary relationship with the description of association in an AODB model.

The multiplicity of the association **A** is exactly one, meaning that each object of collection **X** must be associated with exactly one association, i.e. an object of association **A**. An association-object may be associated with any chosen number of objects of collection **X**. Therefore, there is only one role which determines this relationship and corresponds to association **P** in the OODB model. Roles **R1** and **R2** determined in the OODB model are disregarded in the AODB model, since they always correspond to a relating element and an element being related accordingly, which is automatically ensured by the data storage model.

Association is a role owner. Roles are used to connect objects within a given relationship, what clearly specifies that each object involved in this relationship is assigned a specific role. However, it may be needed to describe relationship itself. One of the solutions would be setting up a new role and connecting collection to it. This way, however, will lead to another semantic ambiguity since the object supposed to describe the association would actually take part of it. In the example (Fig. 7), the collection **GROUP** is responsible for defining of objects' structure that is used to describe the association **Group**. The function of description is not equal to taking part of association on the same conditions as objects of the collection **MEMBER**.

We had to create a binary association with the role **Group** (Fig. 8), the semantics of such a relationship would determine that the objects of collections **GROUP** and **MEMBER** constitute a whole merged by the association **Group**. The meaning of this scheme is different from the intention to present the group as a kind of compound with members participating in it. Objects of collections **GROUP** and **MEMBER** semantics should be different and this distinction is reflected in the model on the level of grammar. This increases the uniqueness of the description of the modeled reality.

Fig. 8. Sample binary relationship with the roles **Member** and **Group** in an AODB model.

4.2. Generalized problems of the binary relationship representation

4.2.1. The relational database model

In the RDB model, a binary relationship is the most natural one from the perspective of physical establishment. Two tables are linked by means of a mechanism of attributes performing the functions of *primary* and *foreign keys*. Figure 9 shows a sample method of establishing a binary relationship in the RDB model. Attribute ID_X in table X is the *primary key*, whereas attribute ID_X in table Y is the *foreign key*. By associating these keys with a relation of equality $X.ID_X = Y.ID_X$, one can establish a binary relationship between tables X and Y, however, the multiplicity of such a relationship is *one-to-many* [12, 13].

The uniqueness of the *primary key* as well as the fact that a *foreign key* is not required ensure establishment of the binary *one-to-many* relationship.

For the purposes of the example in Fig. 9:

- Table X performs a function referred to as a dictionary for table Y;
- table Y contains data referring to the dictionary table X;
- A is a relationship functioning as an equality relation X ;
- attribute ID_X from table X performs the function of the *primary key*; and
- attribute ID_X from table Y performs the function of the *foreign key*.

A binary relationship of *many-to-many* multiplicity established in an RDB model requires an additional table performing a special function, namely linking tuples from other tables. In the RDB model, attributes store only single items of data. Figure 10 illustrates a sample method of establishing a physical binary *many-to-many* relationship in the RDB model.

For the purposes of the example in Fig. 10:

- Tables X and Y store the related data;
- table A establishes the *many-to-many* relationship between tables X and Y;
- attribute ID_X from table X performs the function of the table's *primary key*;

Fig. 9. Sample binary relationship of the *one-to-many* type in an RDB model.

Fig. 10. Sample binary relationship of the *many-to-many* type in an RDB model.

- Even though table **A** is the same as tables **X** and **Y** in the sense of the physical form of the database scheme, it still performs a completely different function. It results from the semantics of relationships in the given scheme as assumed by the designer. Generally speaking, in order to read the meaning and determine the functions of tables present in physical schemes of the RDB model in a completely explicit manner, one must supplement the scheme with information describing the semantics of table relationships and functions of individual attributes. The grammar of the database scheme presented in the RDB model is ambiguous as regards the semantics. Hence the interpretation depends on its context. In semantic terms, table **A** is a completely different entity than tables **X** and **Y**.

In the OODB model, similarly to the RDB model, a binary relationship is the most natural relationship type. One should also note that “The Object Data Standard: ODMG 3.0” [5] assumes binary relationships exclusively, i.e. Sec. 2.6.2 — *Relationship* of the standard reads as follows:

Figure 11 shows a scheme featuring two classes, i.e. X and Y, as well as association A with the multiplicity of *one-to-many*.

- **X** and **Y** are classes storing objects constituting data;
- association **A** has two roles: **R1** and **R2**;
- role **R1** has the multiplicity of 1..1, meaning that objects from class **Y** must be associated with exactly one object of class **X** (by means of association **A** and role **R1** of this association); and

Fig. 12. Sample conceptual binary relationship of the *many-to-many* type in an OODB model.

- role R2 has the multiplicity of 0..*, meaning that objects from class X may be associated with any chosen number of objects of class Y (by means of association A and role R2 of this association).

Figure 12 shows a scheme featuring two classes, X and Y, as well as association A of the *many-to-many* multiplicity.

For the purposes of the example in Fig. 12:

- X and Y are classes storing objects constituting data;
- association A has two roles: R1 and R2;
- role R1 has the multiplicity of 0..*, meaning that objects of class Y may be associated with any chosen number of objects of class X (by means of association A and role R1 of this association); and
- role R2 has the multiplicity of 0..*, meaning that objects of class X may be associated with any chosen number of objects of class Y (by means of association A and role R2 of this association).

Binary relationships of any chosen multiplicity, and those of the *many-to-many* type in particular, are natural in the object-oriented model. It is due to the fact that class attributes in the OODB model are capable of storing numerous values, which means that they may be, for instance, lists with multiple references (e.g. pointers) to other objects. This model is also known of a specific phenomenon which consists in using attributes for both data storing and for performing a completely different function, namely for class associating. It is a problem analogical to the RDB model. Also the manner in which semantics of individual attributes is determined in diagrams illustrating the OODB model schemes is similar to the RDB model. Attributes performing the functions of associations may sometimes be assigned special designations in object diagrams, and they are simply usually not listed among attributes. The OODB grammar does not assume such differentiation, and hence the semantics of individual attributes is not unambiguous. However, such a solution enables easier discrimination between meanings and true functions of individual elements of a scheme illustrating diagram in the OODB model [5, 10, 11].

4.2.3. *The association-oriented database model*

In the AODB model, a binary relationship is a standard *n*-ary relationship, where *n* = 2. Both in conceptual and physical terms, it is the same relationship. It does not

Fig. 13. Sample binary relationship in an AODB model.

Fig. 14. Example of binary relationship in an AODB model.

differ from the other ones, except for the number of roles it features. Figures 13–15 show binary associations in an AODB model.

For the purposes of the example in Fig. 13:

- X and Y are collections storing objects constituting data;
- association A has two roles: R1 and R2;

Fig. 15. Temporal version of binary relationship example with description collection in an AODB model.

- role R1 has the multiplicity of many for the collection, meaning that any chosen number of objects of collection X may participate in the association enabled by this role, and the multiplicity of one for the association, meaning that each object of collection X must participate in exactly one association; and
- role R2 has the multiplicity of 0..1 for the collection, meaning that no more than one object of collection Y may participate in the association enabled by this role, and the multiplicity of many for the association, meaning that each object of collection Y may participate in multiple associations.

Compared to the unary relationship, another role has emerged, thus establishing a binary relationship. The very core of the idea is an association which performs one and only one function. It groups the roles. Objects are involved in associations exactly by means of these roles. According to the assumption made, each element performs a specific and defined function, whereas the semantics of the diagram is to a large extent a consequence of its grammar. There are no elements performing special functions, i.e. those that differ from the ones they have been established for.

4.3. Generalized problems of the n -ary relationship representation

4.3.1. The relational database model

In the RDB model, an n -ary relationship is purely conceptual, i.e. it is impossible to establish directly by means of the categories defined in it [12, 13]. In conceptual terms, nearly everything is possible in every model, but what truly matters is what the model actually supports, namely the physical form of a database scheme. Assuming such a form, in order to establish an n -ary relationship one must first create an auxiliary table analogically to the manner in which the *many-to-many* relationships are established [24]. There is no dedicated element establishing relationships in the RDB model. They are established by imposing specific limitations on the values of keys in tables. The n -arity is ensured by an additional auxiliary table featuring n attributes constituting *foreign keys* to linked tables. As in the cases described above, both tables and attributes perform various roles depending on the context. In more complex diagrams containing schemes following the RDB model, it is not always easy to find tables providing n -ary associations [24]. The difficulty increases due to a situation when, in their surroundings, there are other auxiliary tables performing other special functions, such as the *many-to-many* relationship, for instance. If there is an n -ary *many-to-many* relationship, most tables are auxiliary ones. Figure 16 illustrates a sample n -ary relationship, namely a ternary *one-to-many* relationship in the RDB model, where:

- X, Y and Z are data storage tables;
- A is a table establishing the ternary relationship;
- relationship R1 : $A.ID_X = X.ID_X$;

Fig. 16. Sample *n*-ary (ternary) relationship of the *one-to-many* type in the RDB model.

- relationship R2 : $A.ID_Y = Y.ID_Y$; and
- relationship R3 : $A.ID_Z = Z.ID_Z$.

Figure 16 implies that establishing *n*-ary relationship is relatively simple to accomplish in the RDB model, although it introduces an additional table whose function differs from others. The same situation is not that transparent when the designer applies an *n*-ary relationship simultaneously with the many multiplicities on the data storage tables side. Figure 17 provides a sample *n*-ary relationship (for $n = 3$) for two links of the *one-to-many* multiplicity and one role of the *many-to-many* multiplicity in an RDB model, where:

- X, Y and Z are data storage tables;
- A and XA are tables establishing the ternary relationship;
- relationship R1X : $X.ID_X = XA.ID_X$;
- relationship R1A : $A.ID_A = XA.ID_A$;
- relationship R2 : $A.ID_Y = Y.ID_Y$; and
- relationship R3 : $A.ID_Z = Z.ID_Z$.

Fig. 17. Sample *n*-ary relationship (for $n = 3$) for two roles of the *one-to-many* multiplicity and one role of the *many-to-many* multiplicity in an RDB model.

The *many-to-many* relationship (**R1X** and **R1A**) has been established by means of the **XA** table. A single link may apply to many tuples from table **A** and, at the same time, to many tuples from table **XA**. What matters for the sake of semantic data cohesion is to consider them as a single relationship. With a larger number of links, when the many multiplicity applies, the scheme becomes expanded and its complexity increases. A correct and unambiguous interpretation of such a scheme is becoming difficult and particularly unintuitive. Table **A** performs the role of a *many-to-many* relationship between tables **Y** and **Z**, and so does the **XA** table functioning between tables **X** and **A**. However, the semantics of those two scheme sections is completely different when comparing both cases. With a larger number of *many-to-many* relationships in n -ary relationships (for $n > 3$), the situation is becoming even more complicated, whereas the semantics becomes virtually undeterminable. The semantics does not stem from the database scheme itself, but it must be concealed in database modification procedures, thus being moved from the data layer to the application logic.

4.3.2. The object-oriented database model

In the OODB model, an n -ary relationship is established in the same manner as in the RDB model [18, 21]. It is handled by means of an additional element, i.e. a class performing a linking function [19]. As aforementioned in the section describing binary relationships, the OM ODMG 3.0 Standard does not support n -ary relationships [5, 10, 11]. Figure 18 illustrates a sample ternary relationship in the OODB model, where:

- **X**, **Y** and **Z** are classes storing objects constituting data;
- relationship **A** has three roles: **R1**, **R2** and **R3**;
- role **R1** has the multiplicity of 0..1, meaning that no more than one class **X** object can participate in a relationship;
- role **R2** has the multiplicity of 1..*, meaning that at least one class **Y** object must participate in a relationship; and
- role **R3** has the multiplicity of one, meaning that exactly one class **Z** object participates in a relationship.

Fig. 18. Conceptual n -ary relationship diagram (for $n = 3$) in an OODB model.

Figure 18 implies the number of objects participating in a specific relationship. However, it does not provide any information on the number of relationships in which objects can participate. It is a considerable issue connected with the manner of implementing a relationship. If an association is implemented assuming the form of an additional class, it becomes a set of three binary relationships. They are semantically related which, however, does not result from the physical structure (Fig. 19).

The problem of the multiplicities marked with arrows remains unsolved on the conceptual level. In practice, multiplicities must be determined and they exert a significant impact on the semantics of the given structure. If Fig. 19 illustrated relationships between certain resources, to consider a random example, the multiplicity of many in associations would imply inexhaustible nature of the resource being allocated. Should the said allocation apply to university courses, for instance, then one of groups (e.g. groupZ) might represent a type of courses. Under such conditions, every course would be assigned a type (e.g. lecture, laboratory, project, practicalclasses), but a single object representing this type might participate in any chosen number of specific relationships. For instance, if group Z represented objects constituting a student attendance list for individual courses, then the association multiplicity would have to be 0..1 (*no more than one*).

Similarly to a binary relationship, so the *n*-ary one requires an additional establishing class to be introduced for the sake of its unambiguity in terms of multiplicity. The foregoing means that all relationships should be established by means of an associating class in practice. However, one is still left with the issue of semantics of such a group of associations, namely the lack of an explicit premise enabling the given group to be recognized as a single relationship, and another one as independent associations. Moreover, one still does not know which classes are to be treated as an implementation of relationships, and which as classes of objects participating in the former. Only names of classes may prove useful while attempting to work out an appropriate interpretation method, this hardly to be considered as unambiguous

Fig. 19. Sample physical establishment of an *n*-ary relationship (for *n* = 3) in an object-oriented database model.

semantics. And since some classes in the diagram illustrating an OODB model performs the relationship establishing function, and some describe the objects participating in those relationships, and the groups in question are disjoint, a question arises why the model does not feature a separate category enabling:

- Transparent differentiation between associating classes and classes being associated; and
- grouping of associations for the sake of their semantically unambiguous designation as a single relationship.

In other words, the gist of the problem is to determine where in the scheme the linking elements are to be found and what binary relationships belong to them. These and other issues have been completely eliminated in the AODB.

4.3.3. The association-oriented database model

In the AODB model, an n -ary relationship is established in a natural manner, coherent with the aforementioned unary and binary relationships. Figures 20 and 21 illustrate sample ternary relationship in the AODB, where:

- X, Y and Z are collections storing objects constituting data; and
- association A has three roles: R1, R2 and R3:
 - Role R1 has the multiplicity of 0..1 in terms of the collection, meaning that no more than one object of collection X may participate in the association, and the multiplicity of many in terms of the association, meaning that an object of collection X may participate in any number of association-objects of association A.
 - Role R2 has the multiplicity of 0..1 in terms of the collection, meaning that no more than one object of collection Y may participate in the association, and the multiplicity of many in terms of the association, meaning that an object of collection Y may participate in any number of association-objects of association A.

Fig. 20. Sample n -ary relationship (for $n = 3$) in an AODB model.

Fig. 21. Sample *n*-ary (ternary) relationship in an AODB model.

- Role **R3** has the multiplicity of one in terms of the collection, meaning that exactly one object of collection **Y** must participate in the association, and the multiplicity of many in terms of the association, meaning that an object of collection **Y** may participate in any number of association-objects of association **A**.

In the AODB, an association constitutes a separate model category. Both the object-oriented and the association-oriented diagrams are similar, but their grammatical and semantic differences are very crucial. In the AODB, an association is fit for a single function only, whereas in the OODB, the association function is performed by a class which is to be read in an implicitly grammatical manner, differently than all other classes. In the AODB, roles constitute integral components of associations, and associations exclusively, whereas in the OODB, the equivalents of roles are binary relationships. Their definition spreads across two classes participating in this relationship. The arity of an association in the AODB corresponds to the number of roles it has been assigned. Multiplicities of roles occur for both the collection and the association. Collections do not own roles. From the perspective of collections, a relationship between a collection and an association, referred to as a role, should be considered as the rights of objects to perform a specific role in an association. Roles feature *Navigability* and may be one- (*uniNav*) or two-navigable (*biNav*). In an object-oriented model, arity of relationships is conceptually ambiguous on the scheme grammar level, since one cannot unambiguously determine which associations are to be treated as one relationship. From the point of view of objects of classes, they participate in some relationships, still there is no division into associating objects and those being associated, i.e. both objects are to be treated as equally legitimate elements of the given relationship. The foregoing bears serious consequences in terms of virtual inheritance. Furthermore, in the AODB, owing to the solutions discussed above, there is a possibility of selective inheritance of attributes and roles which means that, while inheriting collections, one may independently inherit attributes and/or roles, or actually the objects' or association-objects' rights

to perform roles in an association. It is a very powerful design mechanism obtained owing to numerous aspects including the introduction of such independent model categories as association and role as well as by releasing an attribute from the obligation to perform an association establishing function. The overall body of inheritance-related problems is extensive and, as such, it exceeds the thematic scope of this paper.

5. Summarization

In the previous section the importance of unary, binary and n -ary relations has been described from the aspects of RDB, OODB and the AODB models. Each of the aforementioned models has its own approach towards this problem, as well as physical solutions for it. Particularly interesting is the approach to the arity of relationships in the context of their cardinality. Furthermore, in the RDB and the OODB models there are no n -ary relationships, and the RDB model does not define *many-to-many* relationships. This raises a number of issues and effects which have been described in the paper and in the following subsections they have been summarized in the context of each arity.

5.1. Summarization for unary relationships

As far as the RDB model is concerned, a relationship is only established artificially. It is accomplished by means of an auxiliary table where only the attributes establishing this relationship are required. Assuming a physical RDB model form, one cannot establish the *many-to-many* relationships, hence this relationship has the multiplicity of one in the table performing the function of a container for relating tuples. The operation of adding or removing a relationship is simple and does not require making any reference to several scheme elements, however, it requires operating with an attribute or attributes performing a special function, namely that of a *foreign key*.

In the OODB model, there are roles which perform technical functions. They associate appropriate objects. It is handled in the same manner as in the RDB model, namely using attributes. Multiplicities of relationships are optional, and hence they constitute no limitation. A relationship is a class which — similarly to tables in the RDB model — features no data storing attributes. Adding or removing a relationship is handled by modifying values of attributes performing special functions, i.e. linking appropriate objects of two classes. All operations modifying relationships are transactions which require an appropriate operation involving one or two attributes (those performing special functions) in one or two classes, all depending on the relationship multiplicity in the relating class.

According to the principles of the AODB model, an association is perceived in the conceptual sense, meaning that it has but a single role (unary relationship). An association stores a list of objects performing a defined role in this association. All modifications to the relationships of objects are introduced by operating with the

association directly. It is enough to perform the add/remove object operation with an association without the necessity of making a reference to attributes of the collection being related. In the AODB model, an association features no attributes, since the data storage model provides no such possibility. The information on the relationship with which the given object is involved is stored in it implicitly. However, if necessary, describing collection may be introduced to hold information about features of the association. The access to this piece of information is only enabled by appropriately implementing the functionalities defined in the data storage model.

The AODB model ensures full compatibility between the conceptual and the physical models. Furthermore, it is transparent in terms of the relationship modification operation. It features no redundant roles and attributes performing any special functions differing from those that they were originally assigned.

5.2. Summarization for binary relationships

As far as the RDB model is concerned, a binary relationship is established using attributes performing special functions, namely those of the *primary key* and the *foreign key*. In order to build such a relationship, one needs an appropriate structure of attributes in all tables. The multiplicity of this relationship is determined on an indirect basis, for instance, by determining the uniqueness of attributes belonging to keys. The model itself does not support this mechanism. One can clearly notice the issue connected with the fact that the same grammatical item performs multiple functions in an implicit manner. This phenomenon is unnatural compared to the way in which men perceive reality. It may also easily lead to modeling errors, since while making a reference to a grammatical element of choice, the scheme designer is each time forced to identify its meaning in the scheme by analyzing the context of its occurrence as well as the semantics left outside the scheme. This effect is particularly transparent while modeling a physical structure of the *many-to-many* relationship in the RDB model.

In the OODB model, the issue of *many-to-many* connections does not occur. Assuming a two-directional navigation in the OM, building relationship depends on attributes on both sides of the association. It should also be stressed that some OODB models (including ODMG 3.0) envisage all relationships being two-directional.

In the AODB model, a binary association is simply a list consisting of two roles. Apart from that aspect, by no means does it differ from associations with the multiplicity of one or *many*. These roles constitute containers for objects participating in the given association (association-object). For the sake of the establishing method, the multiplicity is determined for both the collection and the association. Hence not only is it possible to determine the multiplicity of objects participating in the given association, but also the number of associations in which an object of the given collection may participate. This difference is particularly significant from the modeling perspective. The OODB and RDB models feature no mechanisms enabling

limitations of this type to be established. And it is exactly due to this and other significant differences between AODB model and the remaining ones that one can hardly speak of complete and direct scheme transformation from AODB to others. This problem has been discussed in detail further on.

The RDB model assumes the multiplicity of relationships being established through limitations pertaining to the uniqueness of *primary keys* and expressions linking tables, e.g. $Y.ID_X = A.ID_X$. Hence, one may speak of *one-to-one* and *one-to-many* relationships. The foregoing obviously applies to physical schemes, as conceptual relationships in models may be arbitrary, and therefore it is irrelevant from the perspective of these considerations. What matters is only what can be established in the physical layer. Consequently, one should focus on the object- and association-oriented models. Figure 11 shows the binary relationship being analyzed in the OM.

Figure 22 provides a diagram of objects with mutually exclusive relationships marked. The foregoing results from the fact that the multiplicity of a binary relationship in class X is 1..1. It means that each object of class Y must be related to exactly one object of class X via association A. Figure 22 clearly shows that object Y2 from collection Y is related to two objects from collection X.

Fig. 22. Example of mutually exclusive relationships between objects in an OODB model.

Fig. 23. Sample binary relationship in an AODB model.

Figure 23 depicts an analogical situation analysis for an AODB model.

One should bear in mind that it is the association which features roles in the AODB model. Roles, in turn, are containers for objects in relationships. Hence in order to consider relationships with reference to the AODB model, one should mainly focus on the association. From this perspective, objects of collection X and of collection Y participate in association A. Every type A association-object must be related to exactly one object from collection X and to any chosen number of objects from collection Y. It may be observed that the semantics of this relationship differs from that of the OODB model. Even more considerable differences are taken into consideration when interpreting this relationship from the perspective of collection objects. In the example being discussed, objects of collections X and Y may participate in any chosen number of different association-objects. It means that Fig. 24 drawn up for an AODB model is entirely correct in relation to the intentional diagram provided in Fig. 23.

The foregoing stems from the fact that multiplicities apply not only to collections but also to associations in the AODB model. In the object-oriented model, there is no independent entity of an association. Associations are involved in classes, and their

Fig. 24. Sample extensional diagram in an AODB model.

definition is distributed among all classes participating in them. Therefore, one may claim that associations function in the object-oriented model only conceptually, but no such physical category exists. Consequently, modeling of a situation, where objects of any chosen collection, e.g. Y, may participate in multiple relationships of the given type, but within the given relationship they may only associate with a single object from another collection, e.g. X, would require additional auxiliary classes to be established in the object-oriented model. They would perform associating functions exclusively, or in other words, one would actually be required to model what has been developed in the AODB model by means of the object-oriented one. However, in the AODB model, an association is a dedicated entity grouping roles, hence the OODB model would need to feature a normal class performing different functions than other classes for the sake of expediency. And again, the semantics of such a solution will not be unambiguous, since it would be yet another set of binary relationships whose interpretation depends on the context.

Figure 25 illustrates a method ensuring the functionalities connected with modeling of multiplicities of relationships analogically to the AODB model. However, this solution entails an issue related to the function of associating class A. It is an identical class as all others. Therefore, bearing such a solution in mind, one can hardly speak of a relationship between classes X and Y, all the more about multiplicity of relationships between objects of these classes. In a grammatical sense, this diagram illustrates two separate binary relationships: X–A and Y–A. No relationship between X and Y should be taken into account from the grammatical point of view, and hence the semantics in question is ambiguous and undetermined through the structure. Consequently, the sample diagram of classes in OM still provides no semantics for the intentional diagram developed for AODB. What matters particularly is that a scheme in the AODB model, unlike in the object-oriented one, is semantically unambiguous for the examples discussed. At the same time, it enables determination of the multiplicity of objects with reference to a relationship as well as the number of objects participating in the given association-object. Combining numerous options for establishing the relationship multiplicity in the AODB model, maintaining full unambiguity at the same time, is of major importance for the quality of the process of a guiding idea transformation into a physical form as well as for the subsequent interpretation of intentional schemes and extensional databases.

Fig. 25. Establishing a binary relationship in the OODB model using an additional class performing the associating function.

Fig. 26. Sample binary relationship in an AODB model.

Let us now analyze an example featuring the following semantics. There are three sets: **leaders**, **groups** and **members**. Furthermore, a number of principles pertaining to their mutual relationships are defined as follows: **leader** is a **leader** of at least one **group**, **group** has exactly one **leader** and at least one **member** and **member** belongs to at least one **group**. An intentional database scheme following the principles of the AODB model corresponding to the foregoing is as follows.

Figure 26 illustrates an unambiguous scheme with the semantics described above. It does not require any further descriptions or functionalities in order to perform operations on it. The AODB model ensures that all dependences defined are accomplished. Figure 27 illustrates a binary relationship established without an additional associating class applied. The foregoing scheme does not conform to the aforementioned semantics, since it enables the leader-type relationship to be established unambiguously for a single group only.

Figure 28 depicts a scheme where each **member** is associated with at least one **leader**.

Such a solution is also nonconforming with the requirements of the semantics envisaged, since each **member** may be associated with many **leaders**, but one does not know in what **groups** these relationships occur, whereas the assumption made has been that a **group** has exactly one **leader**, and this cannot be determined in the scheme above.

Figure 29 provides solutions for an object-oriented model entailing an additional class constituting a combination of the **leader** and the **member** class.

Fig. 27. Sample binary relationship with the multiplicity of 1..1 to 1..* in an OODB model without an additional associating class.

Fig. 28. Sample binary relationship with the multiplicity of 1..* to 1..* in an OODB model without an additional associating class.

Fig. 29. Sample binary relationship with the multiplicity set in an OODB model with an additional associating class.

The foregoing scheme complies with the multiplicity-related assumptions adopted for the example analyzed. However, it is also a set of two relationships: **LEADER**–**Group** and **Group**–**MEMBER**. In this respect, one cannot speak of the **LEADER** and **MEMBER** relationship. As far as the binary relationship is concerned, the semantics is relatively easy to guess. It proves particularly helpful in such a case to name classes evocatively. Involving the additional class dedicated to binary relationships is troublesome when a more complex scheme emerges, one which features many relationships of different multiplicities. It is then that unambiguous determination of the semantics is no easy task. Moreover, for instance, in order to remove a chosen **group**, for the sake of the database cohesion one must modify all three objects of three classes. In the **AODB** model, removing a **group** could be brought down to a single operation of the association-object removal. This is also the case when a **leader** or a **member** in the given **group** is to be modified. In the object-oriented model, in each of these cases, one would need to modify two objects belonging to two classes, this being the prerequisite of cohesion. In the **AODB** model, such a modification is brought down to modification of a single object performing a specific role.

Figure 29 depicts a structure similar to the method of establishing a relationship in the **AODB** model. There is but one significant difference between these solutions. It does not directly result from the relationship multiplicity, but rather from the way in which the association is accomplished in the **AODB** model. In the **OODB**, one may attempt to represent an association as an additional class, difference being that, in the **AODB**, relationships are established as roles being parts of the association. In the **OODB** model, a relationship is established with attributes on both sides of the association. It also exerts a considerable impact during inheritance. In the **AODB**, roles are inherited in the course of inheritance of associations. As for collections, one can selectively establish whether attributes or roles are to be inherited. It is also possible to inherit both collection roles and attributes at the same time. What matters in this respect is that, in collections, roles merely determine the right of the given collection to perform roles in the corresponding associations (*Assoc*). The foregoing is particularly important for virtual inheritance in a diamond-shaped structure, since the right to perform roles in an association cannot be duplicated as it would make no sense. It means that the role inheritance in collections can only be performed in a virtual mode. Inheritance in **AODB** is a separate problem with a dedicated article elaborating upon it, hence it has only been mentioned in this paper. What matters particularly from the perspective of a discourse concerning arity of relationships is that representing a relationship in the form of an additional

associating class does not automatically cause that AODB functionalities have been implemented in OODB. This type of mapping is available to a limited degree for simple schemes exclusively which mainly results from the unambiguity of AODB in terms of the guiding idea and the scheme by which it is represented. The same is not ensured by OODB where classes and attributes can perform different functions and only the context proves helpful in determining how they are to be interpreted. In OODB, attributes are used in objects to store data in particular, but they can also become useful while establishing relationships between classes. No such dualisms occur in the AODB model.

5.3. Summarization for *n*-ary relationships

Relationships of arity $n > 2$ does not exist in RDB and OODB models. The concept of *n*-ary relationships can only be obtained by the use of design patterns. This is simply done by introducing as additional table (RDB) or class (OODB) that pursues a relationship. At this point semantics dichotomy reveals relying on the fact that the model categories implementing the function of storing data in this system perform a different function, i.e. they constitute relationship. This causes ambiguity in the interpretation of the scheme, since grammar does not explicitly describe the semantics of its elements.

In contrast, AODB supports relationships of any arity natively. This means that the semantic category association is the owner of roles whose number is not limited. Since association is a separate category in AODB, there are no problems in interpreting the schema. In addition, all operations performed on relationships in AODB are implemented on a single, well-defined element, namely association-object. This way there is no problem in maintaining consistency of database relaying on, for example, the need for manual modification of information stored in the binded or binding object.

In our opinion, there is a way to implement *n*-ary relationships in RDB or OODB in a manner functionally similar to AODB solution. The problem lies in the fact that it will always require using artificial structures referred to as design patterns. The perception of complexity in the context of *n*-ary relationships is natural and deserves a direct representation without having to build it from other elements. What is also crucial is the question of the unambiguity of *n*-ary relationships that are built from other elements. The identification of such constructs is ambiguous, since a set of binary relations cannot always be assigned directly to one and only one *n*-ary relationship, what has been discussed in Secs. 4.3.1 and 4.3.2 of the paper.

6. Discussion

The form of a specific model is influenced by the assumptions made and the guiding ideas followed, the most important of which are the minimalism of model categories, the power of expression and the remoteness between the mental idea and its representation.

Minimization of the number of categories occurring in a database model is sometimes perceived as an elegant solution. A model thus obtained is neat and simple to describe. These are undeniable advantages and they are particularly significant from the perspective of a formal description. In this aspect, an argument that it is not worth creating new model categories as they can be created based on the already existing ones would be difficult to undermine. However, databases are not purely theoretical.

The issue discussed in this section is very important and far more extensive. It pertains to concepts of development. And one of these concepts involves a pursuit of an option to describe anything by means of as few universal model categories as possible. The universalism of model categories is extremely valuable in any theory. But the real question is about the practice and what it is like, both in terms of development of theories and specific solutions.

Whatever has been claimed in this paper may well be fiction, unnecessary embellishment, artificial creation and multiplication of new entities that contribute nothing of value. However, we claim that the discussion of the aforementioned examples is not just art for art's sake completely lacking some deeper meaning. This statement can be supported by numerous arguments. Only one of them has been elaborated in detail in this paper, but it seems to be crucial to settle the matter in question. It all boils down to the remoteness between the mental idea and its representation. Representation of a program in the *Prolog* language using the assembler notation would hardly be recognized as concise and transparent or elegant, since it would involve a minimum number of categories (instruction types) in the notation. The reason for the foregoing seems obvious, for the representation in the assembler is too remote from the concept of logical programs. And *Prolog*, as well as object-oriented programming languages, is perfectly suitable for this purpose. The problem of minimalism occurs on many levels of detail, starting with a general development concept, to end up with problems related to a single programming paradigm, for instance. This can be illustrated by the notion of multi-inheritance. There is a heated dispute on whether or not it should be introduced in object-oriented languages [31–34]. Critics resort to two main arguments claiming that, firstly, multi-inheritance is not absolutely necessary as the same objective can be accomplished otherwise (minimalism), and secondly, that it is a dangerous programming technique (overprotectiveness). The matter of minimalism has already been discussed, whereas overprotectiveness involves an idea of physical elimination of certain possibilities from programming, since there are persons having their own specific reasons to be anxious about that. The bizarreness of such a mode of thinking stems from an attitude of “let us forbid everyone everything that I consider to be dangerous.” It would definitely be enough for all those anxious about multi-inheritance being used in object-oriented programming or object-oriented database modeling to simply refrain from applying this technique. It is not mandatory. The multi-inheritance capacity has been analogically introduced in the AODB model, and since it is not an

obligation to apply it, we cannot conceive a reason for anyone to consider this an issue.

The foregoing remarks are important from the perspective of further descriptions of AODB, and the n -arity of associations in particular, which is due to the fact that this model has not been developed with an intention to introduce absolute minimalism and neither is it overprotective. According to us, the main idea behind its creation was focused on the most significant aspect of reducing the remoteness between a database scheme and the concept being modeled in an analyst's or a designer's mind. For modeling categories should reflect natural categories as perfectly as possible for the sake of perception of the reality fraction being modeled [26].

7. Conclusions

The paper presents results of AODB project in terms of relationships building, i.e. n -ary relationships. Conclusions stated below were made on the base of comparison of RDB, OODB and AODB models.

- (A) The scheme unambiguity, the power of expression and the manner in which a model is implemented are crucial from the perspective of a database model's choice in the process of a database application development. It has been proved that some new solutions in this respect are needed, especially in the aspect of native support of n -ary relationships or *many-to-many* cardinalities of relationships.
- (B) The analysis provided has implied that only the AODB provides direct and explicit support for relationships of any chosen arity on the physical level by means of dedicated categories performing exactly one function in a model. Moreover, the model in question makes it possible to define multiplicity both in terms of the relationship and the collections participating in the former.
- (C) A considerable increase in the power of expression of AODB has been attained by breaking classes down into categories of collection and association. Collections are exclusive owners of attributes, whereas roles are assigned to associations. It has enabled a number of possibilities, including an option of selective inheritance of attributes and roles.

The future work for AODB project includes, but is not limited to:

- (a) Development of dedicated database management system.
- (b) Providing basic and advanced design patterns as tools for database designers.
- (c) Formal definition and forthcoming implementation of query language based on graph schema building.
- (d) Qualitative and quantitative evaluations of AODB in comparison to RDB and OODB.

It is a fact that relational databases are the vast of currently used database systems, but it does not prove it is an excellent and versatile solution. This is

confirmed since more and more corporations and academic centers are searching for and carrying out new solutions for database systems. Even if the specific modeling issues are solvable in existing models, it does not imply that these problems cannot be solved more efficiently, more effectively, in a simpler or more natural way in another model. Secondly, it does not imply that one should not seek alternatives regardless of the prevalence of existing solutions. The best example is the development of programming paradigms and in particular the emergence of object-oriented paradigm.

Presented in the paper unary, binary and n -ary relationships modeling in RDB, OODB and AODB models is an artificial comparison with an emphasis on qualitative differences, in particular, unequivocality. This comparison ought not to be perceived as a total critique of existing models but only to demonstrate the possible ability to solve some of the database modeling issues better.

Appendix A. Model Categories (*Metatypes*)

Categories of the AODB model are generic types (metatypes). The level of metatypes in the model has been divided into main, auxiliary as well as intensional and extensional metatypes. Additionally, the following enumerated metatypes have been defined: navigability (*Navigability*), direction (*Direction*), composition (*Composition*), visibility (*Visibility*), virtuality (*Virtuality*) and data type (*DataType*).

Symbol σ marks complex metatypes of the model. Formal notation $\sigma(Coll)$ is to be interpreted as a “metamodel type (metatype being a structure) — *Coll*,” and hence, for example, $\sigma(BaseNode) = (name : string, abstract : bool)$ means that the following complex model category (structure), *BaseNode*, is a tuple composed of two elements: *name* and *abstract*.

In the intensional aspect, both role owners and their destinations always have full information on mutual associations. The property of navigability only matters in the extensional aspect. A role (*Role*) may be *biNav*, and this is when both the owner of role (*AssocObj*) and its destination (*BaseNode*) have information on mutual associations. For the *uniNav* role, only the owner of role (*AssocObj*) has information on the elements being associated:

$$Navigability = uniNav|biNav. \quad (A.1)$$

The direction is used in the role (*Role*). It enables simple and efficient modeling of directed structures, such as directed graphs or hypergraphs, just to mention some examples:

$$Direction = toOwner|toDest|biDir. \quad (A.2)$$

Composition is a feature and a property of a role (*Role*). It determines mutual dependences between an element's lifetime for the role owner and for destination point *comp* : *Composition*:

$$Composition = inOwner|inDest|inBoth|none. \quad (A.3)$$

The scope of visibility applies to the categories of (*Attr*) and (*Role*) as well as (*CollGenSpec*) and (*AssocGenSpec*). It determines how visible attributes (*Attr*) and roles (*Role*) are as well as define inheritance modes for collections (*Coll*) and associations (*Assoc*).

$$Visibility = public|protected|private. \quad (A.4)$$

Virtuality is a property of the inheritance mode. It determines the manner of handling inheritance^a:

$$Virtuality = natural|virtual|real. \quad (A.5)$$

In a model metastructure, *DataType* is used to define the type of attributes (*Attr*) of a collection (*Coll*):

$$\begin{aligned} DataType = & int8|int16|int32|int64|float|double|bool|ascii|unicode| \\ & date|time|date - time|object|reference|byte| \\ & int8 - vector|int16 - vector|int32 - vector|int64 - vector| \\ & float - vector|double - vector|bool - vector| \\ & ascii - vector|unicode - vector| \\ & date - vector|time - vector|date - time - vector| \\ & object - vector|reference - vector|byte - vector| \\ & ascii - string|unicode - string|byte - array. \end{aligned} \quad (A.6)$$

A database (*Db*) is the very fundamental element of a model. It aggregates collections (*Coll*) and associations (*Assoc*):

$$\sigma(Db) = \left(\begin{array}{l} name : string, \\ colls : \langle Coll \rangle, \\ assocs : \langle Assoc \rangle \end{array} \right). \quad (A.7)$$

An association (*Assoc*) is a key model element, and it is responsible for relationships between data in functional terms:

$$\sigma(Assoc) = \left(\begin{array}{l} name : string, \\ abstract : bool, \\ navigability : Navigability, \\ database : Database, \\ descr : Coll, \\ roles : \langle Role \rangle, \\ gens : \langle AssocGenSpec \rangle, \\ specs : \langle AssocGenSpec \rangle, \\ assocobjs : \langle AssocObj \rangle \end{array} \right). \quad (A.8)$$

A collection is functionally responsible for data storage. In the intensional aspect, the collection (*Coll*) aggregates attributes (*Attr*), i.e. it defines the form of objects

(*Obj*) for which, in this sense, it constitutes a type. In the extensional aspect, the collection (*Coll*) constitutes a container for objects (*Obj*) being kept at the following list: *objs* : $\langle \text{Obj} \rangle$:

$$\sigma(\text{Coll}) = \left(\begin{array}{l} \text{name} : \text{string}, \\ \text{abstract} : \text{bool}, \\ \text{navigability} : \text{Navigability}, \\ \text{database} : \text{Database}, \\ \text{descrroles} : \langle \text{Role} \rangle, \\ \text{descrassoc} : \langle \text{Assoc} \rangle, \\ \text{attrs} : \langle \text{Attr} \rangle, \\ \text{gens} : \langle \text{CollGenSpec} \rangle, \\ \text{spec} : \langle \text{CollGenSpec} \rangle, \\ \text{objs} : \langle \text{Obj} \rangle \end{array} \right). \quad (\text{A.9})$$

A role (*Role*) is a component of an association (*Assoc*). It has a name: *name* : *string*, information on virtuality: *virtual* : *bool*, scope of visibility: *scope* : *Visibility*, definition of navigability: *navigability* : *Navigability*, direction: *dir* : *Direction*, multiplicity (minimum and maximum) for the owner and the destination: *mult*[4] : *int* and information on whether there is a composition on the side of the role owner and on that of the destination: *comp* : *Composition*:

$$\sigma(\text{Role}) = \left(\begin{array}{l} \text{name} : \text{string}, \\ \text{virtual} : \text{bool}, \\ \text{scope} : \text{Visibility}, \\ \text{navigability} : \text{Navigability}, \\ \text{dir} : \text{Direction}, \\ \text{mult}[4] : \text{int}, \\ \text{comp} : \text{Composition}, \\ \text{assoc} : \text{Assoc}, \\ \text{descr} : \text{Coll}, \\ \text{type} : \text{BaseNode} \end{array} \right). \quad (\text{A.10})$$

An attribute (*Attr*) is a component of a collection (*Coll*):

$$\sigma(\text{Attr}) = \left(\begin{array}{l} \text{name} : \text{string}, \\ \text{virtual} : \text{bool}, \\ \text{scope} : \text{Visibility}, \\ \text{count} : \text{int}, \\ \text{type} : \text{DataType}, \\ \text{defaultvalue} : \text{Value}, \\ \text{coll} : \text{Coll} \end{array} \right). \quad (\text{A.11})$$

A base node (*BaseNode*) is an abstract^b structure of auxiliary function. This category has been defined in order to generalize the categories of association (*Assoc*) and collection (*Coll*) for which it is a base structure in the sense of inheritance: $Coll \prec BaseNode$ and $Assoc \prec BaseNode^c$:

$$\sigma(BaseNode) = \left(\begin{array}{l} name : string, \\ abstract : bool, \\ navigability : Navigability \end{array} \right). \quad (A.12)$$

Association generalization–specialization (*AssocGenSpec*) determines the inheritance mode for the association (*Assoc*) with reference to the scope of visibility of roles: *rolescope* : *Visibility* and the rights to fulfill roles *fulfilrolescope* : *Visibility* as well as their virtuality: *rolesvirtual* : *Virtuality*:

$$\sigma(AssocGenSpec) = \left(\begin{array}{l} rolescope : Visibility, \\ rolesvirtual : Virtuality, \\ fulfilrolescope : Visibility, \\ assoc : Assoc \end{array} \right). \quad (A.13)$$

Collection generalization–specialization (*CollGenSpec*) is a structure which determines the inheritance mode for the collection (*Coll*) with reference to the scope of visibility of roles: *attrscope* : *Visibility*, rights to fulfill roles: *fulfilrolescope* : *Visibility* and their virtuality: *attrsvirtual* : *Virtuality*:

$$\sigma(CollGenSpec) = \left(\begin{array}{l} attrscope : Visibility, \\ attrsvirtual : Virtuality, \\ fulfilrolescope : Visibility, \\ coll : Coll \end{array} \right). \quad (A.14)$$

Association-objects (*AssocObj*) are instances of an association (*Assoc*):

$$\sigma(AssocObj) = \left(\begin{array}{l} id : AOID, \\ assoc : Assoc, \\ objdescr : Obj, \\ roles : \langle AssocObjRole \rangle, \\ inroles : \langle AssocObjRole \rangle \end{array} \right). \quad (A.15)$$

An object (*Obj*) constitutes an instance of a collection (*Coll*):

$$\sigma(Obj) = \left(\begin{array}{l} id : OID, \\ values : [Value] \end{array} \right). \quad (A.16)$$

An association-object role (*AssocObjRole*) is a list of elements: *basenodeobjs* : $\langle BaseNodeObj \rangle$.^d They fulfill a single role: *role* : *Role* in the given association (*Assoc*) under a specific association-object: *assocobj* : *AssocObj*:

$$\sigma(AssocObjRole) = \left(\begin{array}{l} role : Role, \\ assocobj : AssocObj, \\ roledescr : Obj, \\ basenodeobjs : \langle BaseNodeObj \rangle \end{array} \right). \quad (A.17)$$

A base node object (*BaseNodeObj*) is an auxiliary abstract structure.^e It functions as a type of element stored at the list of elements of the association-object role (*AssocObjRole*):

$$\sigma(BaseNodeObj) = (). \quad (A.18)$$

A non-navigable association-object (*NonNavAssocObj*) is an auxiliary structure developed to enable creation of instances of associations (*Assoc*) which, compared to (*AssocObj*), will lack the list of *inroles* : $\langle AssocObjRoles \rangle$, and hence also the information on the roles in which they take part:

$$\sigma(NonNavAssocObj) = (). \quad (A.19)$$

Analogically to (*NonNavAssocObj*), a non-navigable object (*NonNavObj*) is an auxiliary structure developed to enable creation of instances of collections (*Coll*) which, compared to (*Obj*), will lack the list of *inroles* : $\langle AssocObjRoles \rangle$, and hence also the information on the roles in which they take part:

$$\sigma(NonNavObj) = (). \quad (A.20)$$

Value is a table of strings of bytes stored in: *data*[] : *Bytes*. This string is marked using a type designated as: *Bytes*.

$$\sigma(Value) = \left(\begin{array}{l} data[] : Bytes, \\ count : int \end{array} \right). \quad (A.21)$$

Appendix B. The Case Study on Association-Oriented Database Metamodel

B.1. Case study specification

As it was stated before, AODB metamodel has been formally defined (AFN_{DB}^{AO}) and then implemented in the form of function libraries. Presentation of implementation is rather hard in such a short paper, since the project technical documentation has hundreds of pages and it is dedicated not only to the topic of this paper, i.e. *n*-ary

relationships modeling. Nevertheless, we introduce the implemented application, being a tool for AML_{DB}^{AO} modeling allowing automatic creation of physical form of database ready to be stored as files structure on the data storage media. The following case study is based on model of university, or to be more specific, the didactic aspect. For the sake of clarity of description, the model has been simplified.

Project assumptions: **Lecture** is a relationship of **lecturer**, **topic**, **content** and **slot** defining time and place of a **lecture**. **Lecture** has exactly one **lecturer**, that in turn can have many **lectures**. Analogically, **lecture** has one **topic**, but there might be many **lectures** for one **topic**. **Lecture** can have many **content**, and a particular **content** can be used in many **lectures**. The **slot** defining time and room of the **lecture** can be assigned to one and only one **lecture**, that can take place in one and only one **time and room slot**. **Lecture** is bound with any number of **equipments**, that can be used by many **lectures**. Time and place **slot** is a junction of many **time slots** but only one **room**. Both **time slot** and a **room** can be used in many **time and room slots**. The **room** is connected with any number of **equipments** but one particular **equipment** piece can be assigned to only one **room**. **Content** is a recursive many-to-many structure, i.e. every content can have any number of other contents in it, but it can also be a part of any number of other contents. Moreover, content is bound to any number of teaching materials, that can be assigned to many contents at the same time.

The diagram of described structure has been prepared in AML_{DB}^{AO} and is showed in Fig. B.1.

Some basic queries in AQL_{DB}^{AO} and C++ code for operations on database will also be presented.

B.2. Administration Panel

Administration Panel is an application useful in graphical and table-based modeling of AODB databases. Figure B.2 presents the graphical modeling main window. The schema diagram of the AODB database can be seen there along with panels used for changing the way the diagram is presented and the logs of committed operations.

Despite the graphical approach, the table-based modeling is supported as well, what has been presented in Fig. B.3.

Each database structure saving operation automatically leads to file creation. In this example, the `Sample1.aodb` is created. It comprises both the database structure and the data.

B.3. Usage examples

One can operate such a database by means of AQL_{DB}^{AO} by embedding it into the C++ programming languages (Listing 1).

Fig. B.1. The diagram describing prepared structure.

Fig. B.2. The main window of AODB Admin Panel.

Fig. B.3. The table-based modeling view of AODB Admin Panel.

```
1 | aodb::core::DATABASE sourceDb;  
2 | sourceDb.LoadDatabase ("Sample1.aodb");  
3 |  
4 | std::string queryText =  
5 |     "QUERY"  
6 |  
7 |     "{ "  
8 |         &COLL LECTURER"  
9 |         INROLE Lecturer"  
10 |         &ASSOC Lecture (-)"  
11 |         { "  
12 |             ROLE Subject &COLL SUBJECT (-) {Item::Name == \"Databases 101\" } "  
13 |             AND "  
14 |             ROLE RoomTimeSlot &ASSOC RoomTimeSlot (-)"  
15 |             ROLE Room &COLL ROOM (-) {Floor == 2}"  
16 |         } "  
17 |     }";  
18 |  
19 | aodb::core::DATABASE targetDb;  
20 | aodb::aql::Query query1 (&sourceDb, queryText, &targetDb);  
21 | int result = query1.execute (&targetDb);
```

Listing 1. Exemplary AQL query.

```
1 |  
2 | aodb::core::DATABASE db;  
3 | db.LoadDatabase ("Sample1.aodb");  
4 | std::string commandText = "obj1 = CREATEOBJ TOPIC { Item::Name = \"Introduction to  
5 |     Database Modeling\" };";  
6 |  
7 | aodb::adl::command cmd (&db, commandText);  
8 | cmd.execute ();  
9 |  
10 | aodb::core::OBJECT *obj1 = cmd.GetVariable ("obj1").GetObject ();  
11 | commandText = "DELETEOBJ TOPIC :obj1";  
12 | aodb::adl::command cmd (&db, commandText);  
13 | cmd.AddParameter ("obj1", obj1->GetId ());
```

Listing 2. Exemplary ADL commands.

Data modification operations can be either executed as an ADL_{DB}^{AO} query or directly by means of the function call from dedicated library. The ADL_{DB}^{AO} instruction can be executed directly from C++ language level (Listing 2).

A code sample for simple database operations has been showed in Listing 3.

```

1  DATABASE db;
2  db.LoadDatabase("Sampledb1.aodb");
3
4  COLLECTION *LECTURER = db.GetColl("LECTURER");
5  COLLECTION *TOPIC = db.GetColl("TOPIC");
6  COLLECTION *SUBJECT = db.GetColl("SUBJECT");
7  COLLECTION *ROOM = db.GetColl("ROOM");
8  COLLECTION *TIMESLOT = db.GetColl("TIMESLOT");
9
10 ASSOCIATION *RoomTimeSlot = db.GetAssoc("RoomTimeSlot");
11 ASSOCIATION *Lecture = db.GetAssoc("Lecture");
12
13 OBJECT* obj_l1 = LECTURER->CreateObj();
14 obj_l1->SetValueByName("Name", VALUE::FromUnicodeString("Walter"));
15 obj_l1->SetValueByName("Surname", VALUE::FromUnicodeString("White"));
16
17 OBJECT *obj_s1 = SUBJECT->CreateObj();
18 obj_s1->SetValueByName("Item::Name",
19                       VALUE::FromUnicodeString("Chemistry"));
20
21 OBJECT *obj_t1 = TOPIC->CreateObj();
22 obj_t1->SetValueByName("Item::Name",
23                       VALUE::FromUnicodeString("Ion dissociation"));
24
25 OBJECT *obj_r1 = ROOM->CreateObj();
26 obj_r1->SetValueByName("Name",
27                       VALUE::FromUnicodeString("Chemistry Room 213"));
28 obj_r1->SetValueByName("Number", VALUE::FromInt16(213));
29 obj_r1->SetValueByName("Symbol", VALUE::FromUnicodeString("CR1"));
30 obj_r1->SetValueByName("Building",
31                       VALUE::FromUnicodeString("Faculty of Chemistry "
32                                                "Building"));
33 obj_r1->SetValueByName("Floor", VALUE::FromInt8(1));
34 obj_r1->SetValueByName("Wing", VALUE::FromUnicodeString("A"));
35
36 OBJECT *obj_ts1 = TIMESLOT->CreateObj();
37 obj_ts1->SetValueByName("Name", VALUE::FromUnicodeString("Ts1"));
38 obj_ts1->SetValueByName("Number", VALUE::FromInt8(1));
39 obj_ts1->SetValueByName("TimeMargins",
40                       VALUE::FromTimeVector(QVector<QTime>()
41                                             << QTime(7,0,0)
42                                             << QTime(9,0,0)));
43
44 OBJECT *obj_ts2 = TIMESLOT->CreateObj();
45 obj_ts1->SetValueByName("Name", VALUE::FromUnicodeString("Ts2"));
46 obj_ts1->SetValueByName("Number", VALUE::FromInt8(2));
47 obj_ts1->SetValueByName("TimeMargins",
48                       VALUE::FromTimeVector(QVector<QTime>()
49                                             << QTime(10,0,0)
50                                             << QTime(12,0,0)));
51
52 obj_l1->SetValueByName("Surname",
53                       VALUE::FromUnicodeString("Heisenberg"));
54
55 ASSOCIATIONOBJECT *aobj_rts1 = RoomTimeSlot->CreateAssocObj();
56 aobj_rts1->GetRole("Room")->AddObj(obj_r1);
57 aobj_rts1->GetRole("TimeSlot")->AddObj(obj_ts1);
58 aobj_rts1->GetRole("TimeSlot")->AddObj(obj_ts2);
59
60 ASSOCIATIONOBJECT *aobj_l1 = Lecture->CreateAssocObj();
61 aobj_l1->GetRole("Lecturer")->AddObj(obj_l1);
62 aobj_l1->GetRole("Topic")->AddObj(obj_t1);
63 aobj_l1->GetRole("Subject")->AddObj(obj_s1);
64 aobj_l1->GetRole("RoomTimeSlot")->AddObj(aobj_rts1);
65
66 TIMESLOT->DeleteObj(obj_ts1);

```

Listing 3. Exemplary database operations.

The case study is but a sample of elements comprising AODB implementation. AODB as a project has number of components that constantly are being enriched and excelled.

References

1. P. P.-S. Chen, The entity–relationship model toward a unified view of data, *ACM Trans. Database Syst.* **1**(1) (1976) 9–36.
2. C. J. Date, *An Introduction to Database Systems*, Vol. 1 (Addison-Wesley, 1995).
3. J. Pokorny, NoSQL databases: A step to database scalability in Web environment, in *Proc. 13th Int. Conf. Information Integration and Web-based Applications and Services*, 2011, pp. 278–283.
4. B. G. Tudorica and C. Bucur, A comparison between several NoSQL databases with comments and notes, in *Proc. 10th RoEduNet Int. Conf.*, 2011, pp. 1–5.
5. M. Berler, J. Eastman, D. Jordan, C. L. Russell, O. Schadow, T. Stanienda and F. Velez, *The Object Data Standard: ODMG 3.0* (Morgan Kaufmann, 2000), p. 280.
6. R. Adamus, P. Habela, K. Kaczmarek, M. Lentner, K. Stencel and K. Subieta, Stack-based architecture and stack-based query language, in *Proc. 1st ICOODB Conf.*, 2008, pp. 77–96.
7. K. Subieta, *Teoria i konstrukcja obiektowych języków zapytań* (PJWSTK, Warsaw, 2004), p. 522.
8. M. Krótkiewicz and K. Wojtkiewicz, Conceptual ontological object knowledge base and language, in *Computer Recognition Systems*, eds. M. Kurzyski, E. Puchała, M. Woźniak and A. Żołnierczyk, Vol. II (Springer-Verlag, 2005), pp. 227–234.
9. M. Krótkiewicz and K. Wojtkiewicz, Functional and structural integration without competence overstepping, *J. Logic Lang. Inf.* **23**(3) (2014) 331–345.
10. M. Weisfeld, *The Object-Oriented Thought Process* (Addison-Wesley Professional, 2003).
11. J. Banerjee, W. Kim, H.-J. Kim and H. F. Korth, Semantics and implementation of schema evolution in object-oriented databases, *ACM SIGMOD Rec.* **16** (1987) 311–322.
12. E. F. Codd, *The Relational Model for Database Management: Version 2*, 2nd edn. (Addison-Wesley, 1990).
13. P. A. Bernstein and E. Newcomer, *Principles of Transaction Processing*, 2nd edn., The Morgan Kaufmann Series in Data Management Systems (Morgan Kaufmann, 2009).
14. S. W. Dietrich and S. D. Urban, *Fundamentals of Object Databases: Object-Oriented and Object-Relational Design* (Morgan and Claypool, 2010).
15. A. Albano, G. Ghelli and R. Orsini, Fibonacci: A programming language for object databases, *VLDB J.* **4**(3) (1995) 403–444.
16. F. Ferrandina and S.-E. Lautemann, An integrated approach to schema evolution for object databases, in *OOIS'96*, eds. D. Patel, Y. Sun and S. Patel (Springer London, 1997), pp. 280–294.
17. P. Hayes, J. Carroll, C. Welty, M. Uschold, B. Vatant, F. Manola, I. Herman and J. Lawrence, Defining N -ary relations on the semantic Web, W3C Working Group Note, 12 April 2006, World Wide Web Consortium (W3C), Cambridge, MA, USA.
18. L. Cerf, J. Besson, C. Robardet and J. F. Boulicaut, Closed patterns meet n -ary relations, *ACM Trans. Knowl. Discov. Data* **3**(1) (2009) 3.
19. G. Génova, J. Llorens and P. Martínez, The meaning of multiplicity of n -ary associations in UML, *Softw. Syst. Model.* **1** (2002) 86–97.
20. F. De Marchi, S. Lopes and J. M. Petit, Unary and n -ary inclusion dependency discovery in relational databases, *J. Intell. Inf. Syst.* **32** (2009) 53–73.

21. L. Cerf, J. Besson, C. Robardet and J.-F. Boulicaut, Data-Peeler: Constraint-based closed pattern mining in n -ary relations, in *Proc. SIAM Int. Conf. Data Mining*, 2008, pp. 37–48.
22. S. Davies, J. Zeitz and J. Hatfield, Addressing the cognitive difficulties of expressing n -ary relations in semantic web data, in *Proc. 6th Int. Conf. Semantic Systems*, 2010, p. 1.
23. P. Severi, J. Fiadeiro and D. Ekserdjian, Guiding the representation of n -ary relations in ontologies through aggregation, generalisation and participation, *J. Web Semant.* **9** (2011) 83–98.
24. C. Soutou, Knowledge discovery in relational database: Extraction of n -ary relationships, in *Proc. Third Basque Int. Workshop on Information Technology*, 1997.
25. S. B. Navathe, Evolution of database modeling for databases, *Commun. ACM* **35**(9) (1992) 112–123.
26. P. Beynondavies and A. R. Hutchings, Modeling and databases, *Eur. J. Oper. Res.* **64**(3) (1993) 327–337.
27. M. Teisseire, P. Poncelet and R. Cicchetti, A tool based on a formal approach for object-oriented database modeling and design, in *Proc. 6th Int. Workshop Computer-Aided Software Engineering*, 1993.
28. B. Catriel, A formal approach to object-oriented databases, *Data Knowl. Eng.* **5**(4) (1990) 353–382.
29. F. Stajano, A gentle introduction to relational and object oriented databases, ORL Technical Report TR-98-2, The Olivetti and Oracle Research Laboratory, Cambridge, UK (1998).
30. P. Habela and K. Subieta, Standard metamodel for object databases: Roles and issues (Extended Abstract), in *Proc. TOOLS'01 Object Databases Workshop*, 2001.
31. D. Malayeri and J. Aldrich, CZ: Multiple inheritance without diamonds, in *Proc. OOPSLA Companion'09*, 2009, pp. 21–40.
32. A. Taivalsaari, On the notion of inheritance, *ACM Comput. Surv.* **28** (1996) 438–479.
33. B. Stroustrup, Multiple inheritance for C, *Comput. Syst.* **2** (1989) 367–395.
34. P. Rong, J. Cui and S. Chen, The implement mechanism and efficiency analysis of multiinheritance and polymorphism, in *Proc. Technology of Object-Oriented Languages*, 1997.