Computer Science and Information Systems 15(1):51–78 https://doi.org/10.2298/CSIS170630046K

A Novel Inheritance Mechanism for Modeling Knowledge Representation Systems

Marek Krótkiewicz

Department of Information Systems,
Wrocław University of Science and Technology,
Wybrzeże Stanisława Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract. The mechanism of inheritance is a powerful tool used to describe complexity of a reality fraction. It is particularly important for knowledge representation systems modelling. It provides a specific ability to take over properties from a base element, what is crucial for conciseness, and modelling efficiency as well. In the Association-Oriented Database Metamodel, inheritance retains coherence with the object-oriented model in the most general terms. However, it has been otherwise defined which stems from the specificity of the metamodel, and particularly from its capabilities which have blazed a trail for its further extension. The main contribution of this article is a description of preliminary assumptions, postulates and conceptual solutions applicable to inheritance. They have been discussed against the background of the Association-Oriented Database Metamodel as well as an object-oriented model compared with the former.

Keywords: knowledge representation, databases, modeling theory, inheritance, Association-Oriented Database Metamodel, Semantic Knowledge Base, object-oriented model.

1. Introduction

This article provides a discussion concerning the mechanism of inheritance in the Association-Oriented Database (AODB) Metamodel [17, 23]. It is one of the key features particularly important in complex knowledge representation systems modeling. [24, 28]. It also addresses an analogical mechanism applied in the *object-oriented metamodel*. It is for the specificity of the AODB Metamodel that inheritance can be defined in a manner far more extensive than in the object-oriented model.

The original motivation to develop the AODB Metamodel was a necessity to build a database management system for a research project referred to as Semantic Knowledge Base (SKB) [25, 26, 27]. It is a project based and focused on concepts, unlike most projects which are focused on terms. A concept is understood as a set of semantic links with other concepts. Consequently, SKB is not a database of linguistic nature, where the main element is a term. It is a multifaceted project featuring numerous dedicated modules. The degree of complexity of data structures is very high, as is the number of mutual associations and dependences of various nature. Therefore, it was necessary to ensure a data storage layer meeting the requirements of SKB. Hence the decision to develop a new database model, since as a result of multiple attempts to apply relational databases,

followed by the object-oriented ones, too many crucial functionalities still remained to lack support from these models. This is how an independent research problem came into being, one which is referred to as AODB Metamodel.

The AODB constitutes an extension of the object-oriented approach. There is currently no single universal standard governing the *object-oriented database model* (*OODB*) [5, 13, 15, 21], which is due to various reasons. The last attempt made to create a standard for object-oriented databases was a project referred to as *ODMG 3.0* [6] developed in 1999 and published in 2000. It contains many compromises and is very general in nature. Moreover, it is difficult to implement, and even impossible to implement in the *Object Query Language* (*OQL*) [1, 7, 14, 16, 19]. Most scientists or database system designers, while creating models or implementing their own solutions, refers to this standard to a greater or lesser extent, mainly in order to highlight differences and to explain reasons for abandoning this standard in favour of one's own concepts. Designers of database management systems focus on ensuring permanence of objects, whereas the approach preferred in terms of the query language is based on extending *SQL* with object-related capabilities or developing *SQL*-like languages. Another important concept of object-oriented databases, one which is definitely worth emphasising at this point, is *Stack Based Approach* (*SBA*) [18, 22, 32, 35]. Both the theoretical grounds and the cohesion of all elements of the solutions developed are unquestionable advantages of this system, both in the theoretical and the implementation aspect. The query language thus created is very distant from the approach known of *OQL* [2] defined in *ODMG 3.0* [6] as well as from other languages based on syntax or the *SQL* concept, which attempt to transform it into an object language.

The solution proposed in this article pertains to a database model different than the relational or the object-oriented one. It is obviously about the AODB Metamodel. In this model, unlike in object-oriented models, it is the association ($Assoc$) that remains in the focus. The scope of competence of this AODB Metamodel category covers the subject of relationships between data. Data containers are collections ($Coll$). The splitting of functions into those related to data storage and those connected with linking them is a key to this model, and particularly to the problem of inheritance.

The section directly connected with the inheritance mechanism in the AODB has been divided into subsections concerning the general concept, the inheritance mechanism description known from object-oriented programming (analysed based on the example of the *C++* language), detailed description of inheritance modes, inheritance algorithms and, last but not least, related work in field of various aspects of inheritance in systems modeling and databases. Moreover, possible application of proposed solution in existing knowledge-based systems has been elaborated. The final part of the article is a summary comprising references to the most important elements of the inheritance mechanism functioning in the AODB Metamodel as well as the conclusions drawn. To present the definition of formal means used, the appendix A containg the formal notation of the AODB Metamodel has been provided.

2. Defining the Research Problem

The most primal and generally formulated research problem taken into consideration is the **development** of a *database model* optimising the following properties:

- *model functionality*, i.e. the available mechanisms and a wide range of solutions in the scope of structure modelling on a minimised set of limitations imposed,
- *development cost*, i.e. the expenditure of resources on the development of modelling tasks, maintained as low as possible,
- *development time*, i.e. the time assumed for the performance of modelling tasks, maintained as short as possible.

As an outcome of the deliberations and analyses conducted, the following postulates have been proposed for the AODB Metamodel: modelling naturalness (intuitiveness), model's power of expression, and semantic unambiguity of schemata.

Category-specific separability of data and relationships between them – The postulate concerning separability of data and the relationships in which the former are involved is about the need for creating at least two separate and completely individual categories. The scope of responsibility of first one would comprise performing a data container function, whereas that of the other one – a data link function. This postulate stems from other postulates, namely the modelling naturalness and the semantic unambiguity of database schemata. Nevertheless, it leads to finding a solution in an association being an independent modelling category.

Single modelling level – It is a solution resulting from the postulate of semantic unambiguity of database schemata. It involves renouncing conceptual categories, i.e. design patterns treated as model elements.

Association as an individual model category – It is a solution stemming from the postulate of category-specific separability of data and relationships between them as well as of *n*-arity of relationships. Association ($Assoc$) is a category responsible for relationships between other model elements.

Possibility of inheriting relationships – It is another outcome of the solution which recognises an association ($Assoc$) as an individual model category. Associations ($Assoc$) may inherit from one another analogically to how it is handled within a collection ($Coll$).

Independence in role and attribute inheritance – It is an outcome derived from the two previously mentioned ones, i.e. a role as an independent category and the possibility of inheriting relationships. This outcome is particularly important, for it enables independent inheritance within data containers and in the scope of relationships.

"Real" inheritance mode – It is a derivative outcome of a group of solutions and other outcomes. The "real" mode removes the virtuality status from the components inherited.

3. Inheritance

3.1. General Concept

Since attributes have been unburdened of the duality consisting in performance of two functions, i.e. containing values and building relationships, it is now possible to independently approach inheritance of attributes and relationships between collections in the AODB Metamodel. Figure 1 provides a sample diagram developed in Association-Oriented Modeling Language (AML^{AO}_{DB})[1].

[1] The syntax of AML^{AO}_{DB} is similar to *UML*® and should be comprehensible for most readers. However etailed description of its syntax and semantics has been described in papers [17, 23, 28].

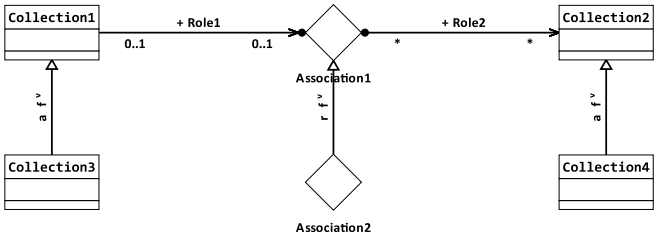


Fig. 1. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting collection ($Coll$) and association ($Assoc$) inheritance in the AODB Metamodel

It comprises three relationships of inheritance. Two of them are related to inheritance of collections, and one to inheritance of associations (1).

$$\begin{gathered} Collection_3 \xrightarrow{a,f^v} Collection_1; Collection_4 \xrightarrow{a,f^v} Collection_2 \\ Association_2 \xrightarrow{r,f^v} Association_1 \end{gathered} \tag{1}$$

Figure 2 provides a diagram illustrating the following roles: $+Association_1 :: Role_1$ and $+Association_1 :: Role_2$, which have been inherited by $Association_2$.

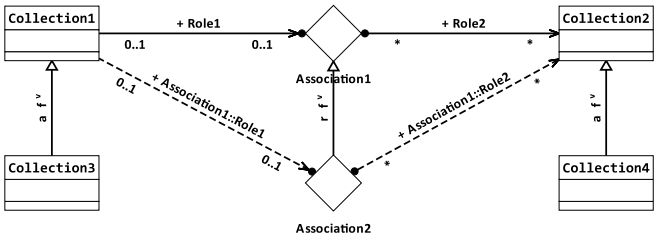


Fig. 2. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting collection ($Coll$) and association ($Assoc$) inheritance in the AODB Metamodel, showing the roles ($Role$) inherited

As a result of the inheritance defined in expression (1), association $Association_2$ has become an owner of the roles ($Role$) inherited from $Association_1$ according to expression (2).

$$Association_2 \left\langle \begin{matrix} [0..1] & \xleftarrow{+Association_1::Role_1} & [0..1] & \square Collection_1, \\ [*] & \xrightarrow{+Association_1::Role_2} & [*] & \square Collection_2 \end{matrix} \right\rangle \tag{2}$$

Figure 3 is a diagram illustrating rights of collection $Collection_3$ to fulfil roles $+Role_1$ and $+Association_1 :: Role_1$ as well as rights of collection $Collection_4$ to fulfil roles $+Role_2$ and $+Association_1 :: Role_2$.

With regard to the object-oriented model, one cannot speak of direct inheritance of relationships between classes. However, there is inheritance of classes, and hence the automatic inheritance of relationships. In this manner, also relationships are inherited

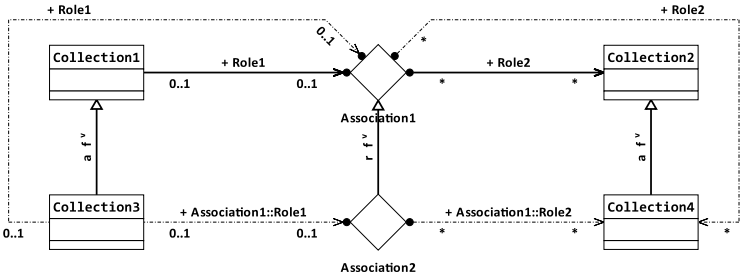


Fig. 3. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting collection ($Coll$) and association ($Assoc$) inheritance in the AODB Metamodel, showing the rights to fulfil roles ($Role$) inherited by a collection ($Coll$)

indirectly and, to a certain extent, "as the opportunity occurs". However, in order that a two-directional relationship could be inherited, it is necessary to conduct inheritance of two classes on both sides of the association.

The above examples show that the case of the AODB Metamodel is completely different, as it lacks the constraints of the object-oriented model. Separating the data combining function from that of data containers has enabled natural and accurate modelling of schemata having a simple structure and relatively high power of expression. This is clearly evident in Figure 4, being a complete diagram comprising both inherited roles and inherited rights to fulfil roles.

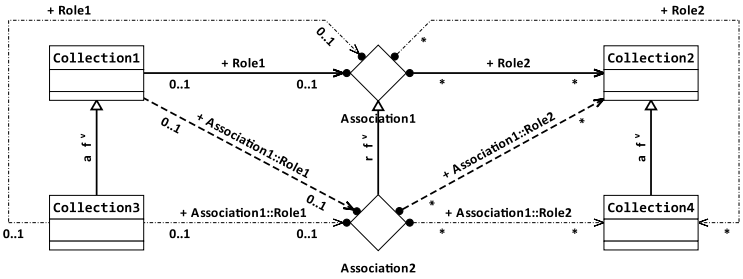


Fig. 4. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting collection ($Coll$) and association ($Assoc$) inheritance in the AODB Metamodel, showing roles ($Role$) and rights to fulfil roles ($Role$) inherited by a collection ($Coll$)

The example 1 shows simple real world scenario of association-oriented model, which depicts the inheritance of collections as well as the associations.

Example 1. Let us consider the following reality: there are two types of *Living*: *Plants* and *Animals*.

$$\square Plant \xrightarrow{a,f^v} \square\ Living; \square Animal \xrightarrow{a,f^v} \square\ Living; \tag{3}$$

The *Livings* are distinguished by the association, which describes their *EatingStrategy*. The *EatingStrategy* comprises role: *EatingStrategy::Eater*. There are three specializations of the strategy: *Herbivorous*, *Carnivorous* and *Omnivorous*,

$$\Diamond Herbivorous \xrightarrow{r,f^v} \Diamond\ EatingStrategy; \Diamond Carnivorous \xrightarrow{r,f^v} \Diamond\ EatingStrategy; \\ \Diamond Omnivorous \xrightarrow{r,f^v} \Diamond\ EatingStrategy \tag{4}$$

which comprise inherited role *EatingStrategy::Eater* and role *Eaten*. Each of them associates the eater *Animal* instance with set of objects of different type (Fig. 5).

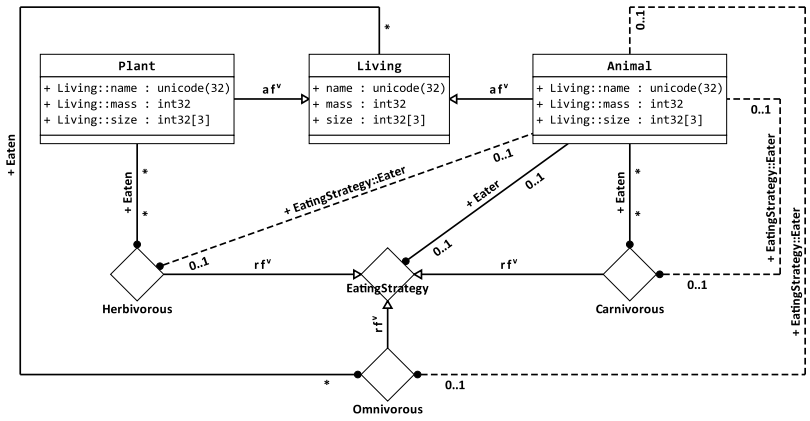


Fig. 5. Diagram developed in $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting model described in Example 1

3.2. Inheritance Mechanism Against the *C++* Language Standard

The *C++* programming language standard defines the inheritance mechanism in a manner depicted in Figure 6.

The inheritance algorithm functioning in *C++* displays nearly no differences in the case of single as well as multiple inheritance. Although both instances are distinguished in the literature of the subject (see e.g. [5, 33]), the actual discrepancies between them are inconsiderable. Consequently, one can easily generalise both cases and apply a single algorithm without losing the very essence of the problem. This algorithm features several crucial elements, all of which have been described below.

- The first important algorithm element is iteration over pre-set base classes.
- When the base class is inherited non-virtually, attributes are inherited, whereas class methods are translated into corresponding functions.
- A table of virtual functions (vtbl) is created next, and the iteration continues over subsequent virtual functions. The structure containing a pointer at the virtual function differs between single and multiple inheritance. Depending on how the virtual

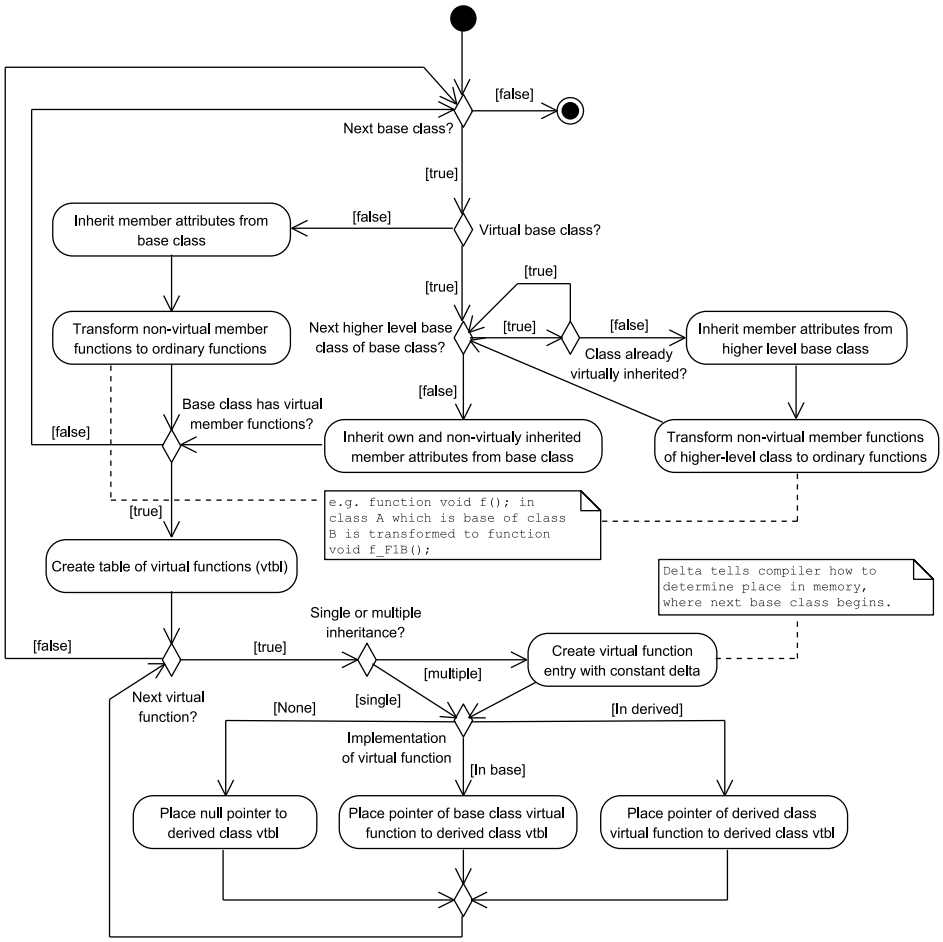


Fig. 6. Algorithm describing the inheritance mechanism in the *C++* language

function has been implemented, what is contained is a pointer at a base class method re-implemented in the derived class, or in the case of a pure virtual function.

– When the base class is inherited virtually, an additional iteration takes place over its base classes (higher-rank base classes). If any of the base classes has already been virtually inherited, it is not inherited again, whereas otherwise the inheritance is handled analogically to the previous case described.

3.3. Inheritance Modes in the Association-oriented Metamodel

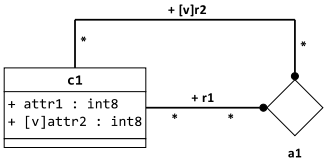
In the AODB Metamodel, the following three inheritance modes have been defined: *n* – natural, *v* – virtual, *r* – real. Natural inheritance, in this case, is exactly what non-virtual inheritance is in the *object-oriented model*. The name has been changed, compared to the *object-oriented model*, in order to distinguish more than two cases which may occur, namely virtual and non-virtual inheritance.

Table 1. Collation of attribute inheritance modes

C++	AODB
non-virual	*n*–natural: $\xrightarrow{a}$
virual	*v*–virual $\xrightarrow{a^v}$
×	*r*–real $\xrightarrow{a^r}$

Virtual inheritance functions analogically in the *object-oriented model* as well as in AODB. In short, in the AODB Metamodel, virtuality of attributes and roles boils down to not duplicating attributes and roles when they originate from multiple inheritance with a shared starting point. Virtuality is a property of an attribute or a role. The idea of virtual elements consists in avoiding a situation when an element (attribute or role) would appear in more than one copy in an association ($Assoc$) or a collection ($Coll$) due to multiple inheritance.

A component, i.e. an attribute ($Attr$) or a role ($Role$), may be assigned the virtual status instantly after being created. Such a case has been illustrated in Figure 7.The diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language has been expressed with formula (5) in the formal notation.



$$a_1 \left\langle \begin{array}{l} [*] \overset{+r_1}{\bullet\!\!-\!\!-} [*] \,\square c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \,\square c_1 \end{array} \right\rangle \qquad c_1 \left\langle \begin{array}{l} +attr_1 : int8, \\ +attr_2^v : int8 \end{array} \right\rangle \tag{5}$$

Fig. 7. Diagram developped in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating virtual attributes and roles in the AODB Metamodel

Attribute inheritance in the virtual mode A database schema, defined by expression (6), whose diagram has been provided in Figure 8, contains four collections ($Coll$) containing an identical set of attributes ($Attr$). The first one – $+attr_1$ – is non-virtual, whereas the second one – $+attr_2^v$ – virtual. What has been taken advantage of at this point is the option of defining components as virtual when they are created.

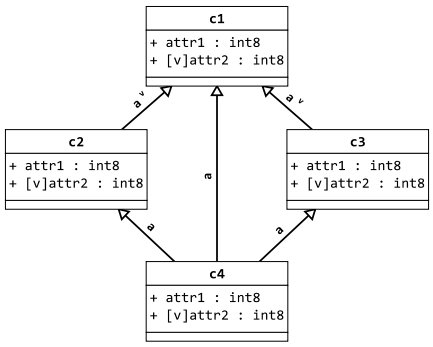


Fig. 8. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating attribute ($Attr$) inheritance in the AODB Metamodel when disregarding inherited attributes

$$\begin{gathered}\{c_1,c_2,c_3,c_4\}\left\langle\begin{array}{l}+attr_1:int8,\\ +attr_2^v:int8\end{array}\right\rangle\\ \square\{c_2,c_3\}\xrightarrow{a^v}\square\, c_1\\ \square c_4\xrightarrow{a}\square\{c_1,c_2,c_3\}\end{gathered}\tag{6}$$

$$c_4\left\langle\begin{array}{l}+attr_1 \quad :int8,\\ +attr_2^v \quad :int8,\\ +c_2::attr_1:int8,\\ +c_2::attr_2^v:int8,\\ +c_1::attr_1^v:int8,\\ +c_1::attr_2^v:int8,\\ +c_3::attr_1:int8,\\ +c_3::attr_2^v:int8,\\ +c_1::attr_1:int8\end{array}\right\rangle\tag{7}$$

As a result of the inheritance, collection ($Coll$) c_4 becomes an owner of attributes defined by expression (7). Attributes $attr_1:int8$ and $+attr_2^v:int8$ are own attributes of collection ($Coll$) c_4. Attribute $+c_1::attr_1:int8$ has emerged in collection c_4 a consequence of inheritance $\square c_4\xrightarrow{a}\square\, c_1$.

$$\begin{gathered}\square c_4\xrightarrow{a}\square\, c_1\vDash c_4\left\langle+c_1::attr_1:int8\right\rangle\\ \square c_4\xrightarrow{a}\square\, c_2\vDash c_4\left\langle+c_2::attr_1:int8\right\rangle\\ \square c_4\xrightarrow{a}\square\, c_3\vDash c_4\left\langle+c_3::attr_1:int8\right\rangle\end{gathered}\tag{8}$$

$$\square c_4\xrightarrow{a}\square\, c_2\wedge c_2\left\langle+attr_2^v:int8\right\rangle\vDash c_4\left\langle+c_2::attr_2^v:int8\right\rangle\tag{9}$$

Attribute $+attr_2^v:int8$ from collection c_2 is inherited by collection c_4 according to formula (9).

As a result of inheritance $\square c_2\xrightarrow{a^v}\square\, c_1$, attribute $+attr_1:int8$ becomes a virtual attribute from collection c_2, and inheritance $\square c_3\xrightarrow{a^v}\square\, c_1$ proceeds analogically, but it applies to collection c_3. Consequently, i.e. as a result of the following inheritances: $\square c_4\xrightarrow{a}\square\, c_2$ and $\square c_4\xrightarrow{a}\square\, c_3$, which matters particularly, it appears in collection c_4 in one item only, namely as $+c_1::attr_1^v:int8$, and not as two attributes, i.e. $+c_2::c_1::attr_1^v:int8$ and $+c_3::c_1::attr_1^v:int8$. This is the very essence of inheritance in the AODB Metamodel. The foregoing dependences are formally expressed in formula (10).

$$\begin{gathered}\square c_2\xrightarrow{a^v}\square\, c_1\wedge c_1\left\langle+attr_1:int8\right\rangle\vDash c_2\left\langle+c_1::attr_1^v:int8\right\rangle;\\ \square c_3\xrightarrow{a^v}\square\, c_1\wedge c_1\left\langle+attr_1:int8\right\rangle\vDash c_3\left\langle+c_1::attr_1^v:int8\right\rangle;\\ \left(\begin{array}{l}\square c_4\xrightarrow{a}\square\, c_2\wedge c_2\left\langle+c_1::attr_1^v:int8\right\rangle\vee\\ \square c_4\xrightarrow{a}\square\, c_3\wedge c_3\left\langle+c_1::attr_1^v:int8\right\rangle\end{array}\right)\vDash\\ c_4\left\langle+c_1::attr_1^v:int8\right\rangle;\end{gathered}\tag{10}$$

An entirely analogical mechanism applies to attribute $+attr_2^v:int8$ from collection c_1, which has been shown in formula (11). The fact that it is a virtual attribute already when created does not change anything in the case analysed.

$$\begin{gathered}\Box c_2 \xrightarrow{a^v} \Box\ c_1 \wedge c_1 \left\langle +attr_2^v : int8 \right\rangle \vDash c_2 \left\langle +c_1 :: attr_2^v : int8 \right\rangle; \\ \Box c_3 \xrightarrow{a^v} \Box\ c_1 \wedge c_1 \left\langle +attr_2^v : int8 \right\rangle \vDash c_3 \left\langle +c_1 :: attr_2^v : int8 \right\rangle; \\ \left(\begin{array}{l} \Box c_4 \xrightarrow{a} \Box\ c_2 \wedge c_2 \left\langle +c_1 :: attr_2^v : int8 \right\rangle \vee \\ \Box c_4 \xrightarrow{a} \Box\ c_3 \wedge c_3 \left\langle +c_1 :: attr_2^v : int8 \right\rangle \end{array} \right) \vDash \\ c_4 \left\langle +c_1 :: attr_2^v : int8 \right\rangle;\end{gathered} \tag{11}$$

Attribute $+attr_2^v : int8$ from collection c_3 is inherited by collection c_4 according to formula (12).

$$\Box c_4 \xrightarrow{a} \Box\ c_3 \wedge c_3 \left\langle +attr_2^v : int8 \right\rangle \vDash c_4 \left\langle +c_3 :: attr_2^v : int8 \right\rangle \tag{12}$$

Figure 9 provides a full picture of inheritance in the form of a diagram depicting inherited attributes ($Attr$) in individual collections ($Coll$).

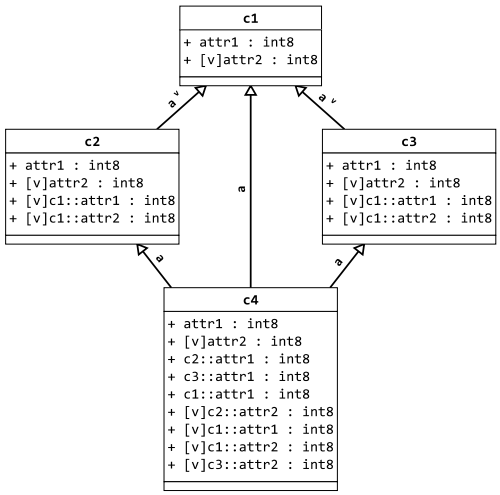


Fig. 9. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating attribute ($Attr$) inheritance in the AODB Metamodel and showing inherited attributes

Example 2. Let us consider the following reality. There is an abstract type of research institution employee, called *Academic*. The Academic is a generalization of two concrete types, such as *Researcher* and *Teacher*. However, some of the academics may be both at the same time. We created the collections for all the aforementioned types and defined corresponding generalizations. To prevent the duplication of attributes, the attributes are inherited in *virtual mode*.

$$\begin{gathered}\Box Researcher \xrightarrow{a^v} \Box\ Academic^{\varnothing}; \Box Teacher \xrightarrow{a^v} \Box\ Academic^{\varnothing}; \\ \Box ResearcherTeacher \xrightarrow{a} \Box\ Researcher \vee \Box ResearcherTeacher \xrightarrow{a} \Box\ Teacher\end{gathered} \tag{13}$$

Consequently, the *ResearcherTeacher* collection obtained attributes $+Academic :: name^v$ and $+Academic :: surname^v$ only once, as shown in the Figure 10.

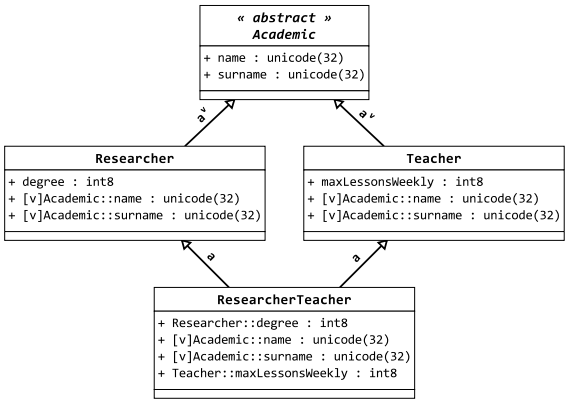
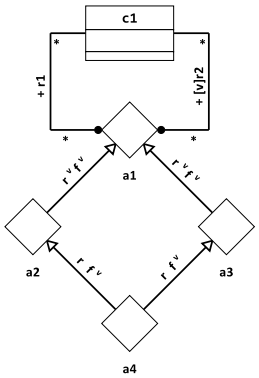


Fig. 10. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating the Example 2

Inheritance of roles The inheritance of roles ($Role$) by associations ($Assoc$) follows identical rules as the inheritance of attributes ($Attr$) by collections ($Coll$). Therefore, only a simple example has been discussed hereafter. A sample database (Db) schema is provided in expression (14). Figure 11 shows a database (Db) schema diagram noted in expression (14).

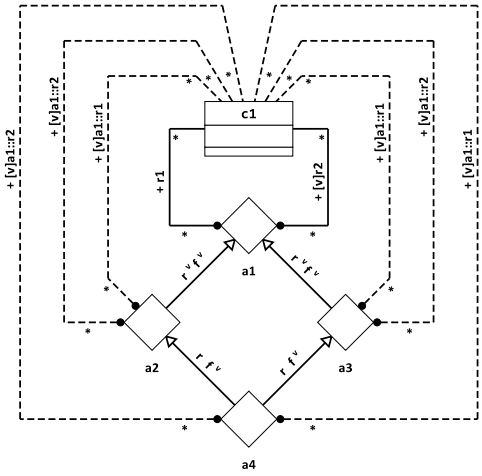


$$a_1 \left\langle \begin{array}{l} [*] \overset{+r_1}{\bullet\!\!-\!\!-} [*] \square\, c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \square\, c_1 \end{array} \right\rangle \qquad (14)$$
$$\diamondsuit \{a_2, a_3\} \xrightarrow{r^v, f^v} \diamondsuit\, a_1$$
$$\diamondsuit a_4 \xrightarrow{r, f^v} \diamondsuit \{a_2, a_3\}$$

Fig. 11. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting role ($Role$) inheritance in the AODB Meta-model

As a result of inheritance, associations a_2, a_3, a_4 obtain the same roles: $[*] \overset{+r_1^v}{\bullet\!\!-\!\!-} [*] \square c_1$ and $[*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \square c_1$, which has been formally noted in expression (15).

Figure 12 provides a database (Db) schema diagram showing the roles inherited in accordance with expression (15).



$$\left(a_1 \left\langle \begin{array}{l} [*] \overset{+r_1}{\bullet\!\!-\!\!-} [*] \square c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \square c_1 \end{array} \right\rangle \wedge \begin{array}{l} \Diamond \{a_2, a_3\} \xrightarrow{r^v, f^v} \Diamond a_1; \\ \Diamond a_4 \xrightarrow{r, f^v} \Diamond \{a_2, a_3\} \end{array} \right) \models \{a_2, a_3, a_4\} \left\langle \begin{array}{l} [*] \overset{+r_1^v}{\bullet\!\!-\!\!-} [*] \square c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \square c_1 \end{array} \right\rangle \tag{15}$$

Fig. 12. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, depicting role ($Role$) inheritance in the AODB Metamodel

Inheritance of rights to fulfil roles Roles ($Role$) belong to associations ($Assoc$), and so they can only be inherited by the latter. Inheritance of rights to fulfil roles applies to both associations ($Assoc$) and collections ($Coll$), and may only proceed in the virtual mode – $\xrightarrow{f^v}$. A different type of inheritance would make no sense, since non-virtual inheritance of rights to fulfil roles would imply a possibility of duplicating two identical rights. When objects of any given collection ($Coll$) or association ($Assoc$) have such a right, then it is in fact irrelevant how many duplicates of this right exist.

A sample database (Db) structure, containing an association ($Assoc$) with two roles ($Role$) and two collections ($Coll$), is represented by expression (16).

As a result of inheritance $\square c_2 \xrightarrow{f^v} \square\, c_1$, collection ($Coll$) named c_2 has obtained the right to participate in two roles: $\overset{r_1}{\bullet\!\!-\!\!-}$ and $\overset{r_2^v}{\bullet\!\!-\!\!-}$. None of the roles ($Role$) has been inherited, hence the only effect attained is that association a_1 may not only connect with with collection ($Coll$) named c_1, but also with collection ($Coll$) named c_2 by means of roles $\overset{r_1}{\bullet\!\!-\!\!-}$ and $\overset{r_2^v}{\bullet\!\!-\!\!-}$. Figure 13 provides a diagram illustrating this situation.

$$a_1 \left\langle \begin{array}{l} [*] \overset{+r_1}{\bullet\!\!-\!\!-} [*] \,\square c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \,\square c_1 \end{array} \right\rangle \quad c_1 \langle\rangle \quad \square c_2 \left\{ \begin{array}{c} \langle\rangle\,; \\ \xrightarrow{f^v} \square\, c_1 \end{array} \right\} \tag{16}$$

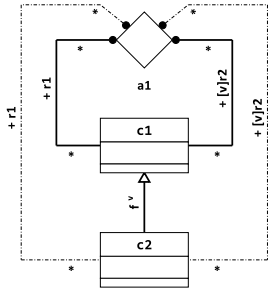


Fig. 13. Diagram developed in the $\mathsf{AML}_{\mathsf{DB}}^{\mathsf{AO}}$ language, depicting inheritance of roles ($Role$) in a collection ($Coll$) in the AODB Metamodel

Principles of inheriting rights to fulfil roles are analogical to those governing inheritance of attributes ($Attr$), subject to a reservation that, as aforementioned, it may only be performed in the virtual mode. The following example illustrates the subject of inheritance of rights to fulfil roles in a diamond arrangement. With regard to the foregoing, a database (Db) represented by expression (17) has been modelled.

As shown in Figure 14, the rights to fulfil roles r_1 and r_2 occurred in all collections which participated in the inheritance, which may be formally noted in accordance with expression (18).

Figure 14 illustrates the foregoing dependences in the $\mathsf{AML}_{\mathsf{DB}}^{\mathsf{AO}}$ language.

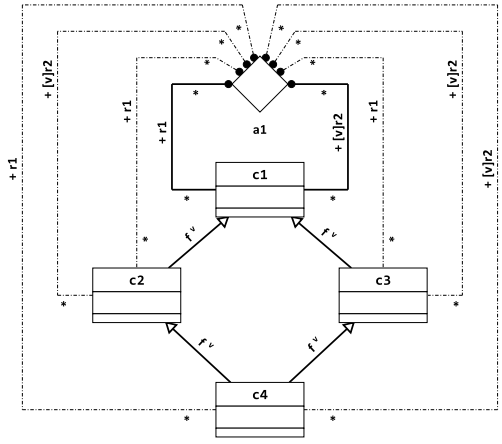


Fig. 14. Diagram developed in the $\mathsf{AML}_{\mathsf{DB}}^{\mathsf{AO}}$ language, illustrating multiple inheritance of rights to fulfil roles by collections ($Coll$) in the AODB Metamodel

$$a_1 \left\langle \begin{array}{l} [*] \overset{+r_1}{\bullet\!\!-\!\!-} [*] \,\square c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \,\square c_1 \end{array} \right\rangle \quad c_1 \langle\rangle\,; c_2 \langle\rangle\,; c_3 \langle\rangle\,; c_4 \langle\rangle \quad \square \{c_2, c_3\} \xrightarrow{f^v} \square\, c_1 \quad \square c_4 \xrightarrow{f^v} \square \{c_2, c_3\} \tag{17}$$

$$\left(a_1 \left\langle \begin{array}{l} [*] \overset{+r_1}{\bullet\!\!-\!\!-} [*] \,\square c_1, \\ [*] \overset{+r_2^v}{\bullet\!\!-\!\!-} [*] \,\square c_1 \end{array} \right\rangle \wedge \square \{c_2, c_3\} \xrightarrow{f^v} \square\, c_1; \; \square c_4 \xrightarrow{f^v} \square \{c_2, c_3\} \right) \models a_1 \left\langle \begin{array}{l} [*] \overset{+r_1(\square\{c_1,c_2,c_3,c_4\})}{\bullet\!\!-\!\!-} [*] \,\square c_1, \\ [*] \overset{+r_2^v(\square\{c_1,c_2,c_3,c_4\})}{\bullet\!\!-\!\!-} [*] \,\square c_1 \end{array} \right\rangle \tag{18}$$

Nevertheless, one should note that the right to fulfil a role resulting from inheritance in the virtual mode (as it is not otherwise possible in the AODB Metamodel) does not become a right to fulfil a virtual role. As for the association ($Assoc$), the roles ($Role$) inherited in the virtual mode become or remain virtual, however, in the case of inheritance of rights to fulfil roles, this is not the case, the reason for which is fairly simple. Since rights to fulfil roles are inherited in the virtual mode only, it does not matter whether or not they have the status of virtuality. In the case of multiple inheritance, a collection ($Coll$) or an association ($Assoc$) may be vested only one right to fulfil a specific role ($Role$), and it does not matter if this right is assigned the status of virtuality, or not. In Figure 14, rights to fulfil roles are marked in a manner implying whether rights to fulfil virtual roles (e.g. +[v]r2) or non-virtual roles (e.g. +r1) have been inherited. This is only the case when the role ($Role$) has such a status. However, even if no such information was available, it would not cause the database (Db) schema to lose unambiguity.

Example 3. In this example, the reality from Example 1 is extended by another 3 generalization relationships. In this case, the collection ($Coll$) *Animal* has three specializations: *Mammal*, *Bird* and *Fish*, which inherit the rights to fulfil role ($Role$) $+Eater$.

$$\square\{Mammal, Bird, Fish\} \xrightarrow{a, f^v} \square\, Animal \tag{19}$$

Consequently, there is only one role ($Role$) $+Eater$ in the model, but it can be fulfilled by the whole collection ($Coll$) taxonomy (i.e. *Animal*, *Mammal*, *Bird*, *Fish* collections). The model has been shown in the Figure 15.

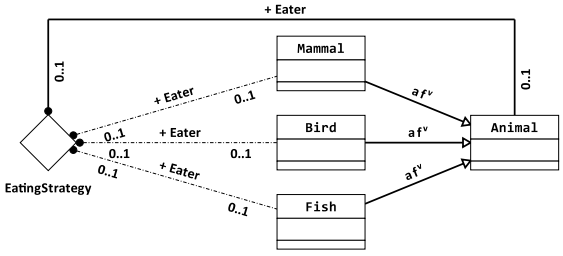


Fig. 15. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating the model from Example 3

Inheritance in the real mode The real inheritance mode complements the inheritance mechanism known from the *object-oriented model*. Virtual inheritance is possible in the *object-oriented model*, which causes that a specific component is assigned the virtual status. However, there is no option of removing such a status, i.e. restoring the non-virtual state, in other words. In the AODB Metamodel, such an operation is possible owing to the real mode. The manner in which this mode functions has been discussed with reference to a sample database (Db) whose schema is provided by expression (20).

$$\{c_1, c_2\}\langle +attr_1 : int8, +attr_2^v : int8\rangle; a_1\left\langle [*] \overset{+r_1}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1, [*] \overset{+r_2^v}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1 \right\rangle; \quad \square c_2 \xrightarrow{a^r, f^v} \square c_1; \Diamond a_2 \xrightarrow{r^r, f^v} \Diamond a_1 \tag{20}$$

Figure 16 provides a diagram corresponding to the schema from expression (20).

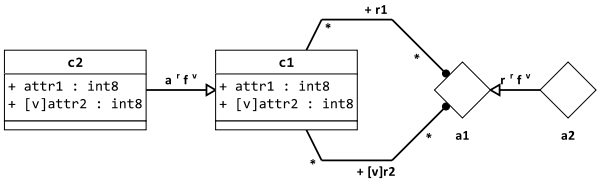


Fig. 16. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating real mode inheritance in the AODB Metamodel

The inheritance mechanism for schema (20) shown on Fig. 16 is described by expression (21).

$$\left(\begin{array}{l} \{c_1, c_2\}\langle +attr_1 : int8, +attr_2^v : int8\rangle \wedge \\ a_1\left\langle [*] \overset{+r_1}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1, [*] \overset{+r_2^v}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1 \right\rangle \wedge \\ \square c_2 \xrightarrow{a^r, f^v} \square c_1 \wedge \Diamond a_2 \xrightarrow{r^r, f^v} \Diamond a_1 \end{array}\right) \vDash$$

$$\left(\begin{array}{l} c_1\left\langle \begin{array}{l} +attr_1 : int8, \\ +attr_2^v : int8 \end{array}\right\rangle; \\ c_2\left\langle \begin{array}{l} +attr_1 : int8, \\ +attr_2^v : int8, \\ +c_1 :: attr_1 : int8, \\ +c_1 :: attr_2 : int8 \end{array}\right\rangle; \\ a_1\left\langle \begin{array}{l} [*] \overset{+r_1(\square\{c_1,c_2\})}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1, \\ [*] \overset{+r_2^v(\square\{c_1,c_2\})}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1 \end{array}\right\rangle; \\ a_2\left\langle \begin{array}{l} [*] \overset{+a_1::r_1(\square\{c_1,c_2\})}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1, \\ [*] \overset{+a_1::r_2(\square\{c_1,c_2\})}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1 \end{array}\right\rangle \end{array}\right) \tag{21}$$

In accordance with expression (21) as well as the diagram provided in Figure 17, as a result of inheritance in the real mode, attribute $c_2\langle +c_1 :: attr_2 : int8\rangle$ does not have the virtuality status, even though it has been declared as $c_1\langle +attr_2^v : int8\rangle$. Analogically, role $a_2\left\langle [*] \overset{+a_1::r_2}{\bullet\!\!-\!\!-\!\!-} [*] \square c_1 \right\rangle$ has lost the status of virtuality as compared with the precursor.

Example 4. Let us assume the following reality. We have set of animals, which are distinguished (among others) by their velocity. Therefore we created the collection ($Coll$) *Animal*. We have two abstract specializations of the animal distinguishing them because

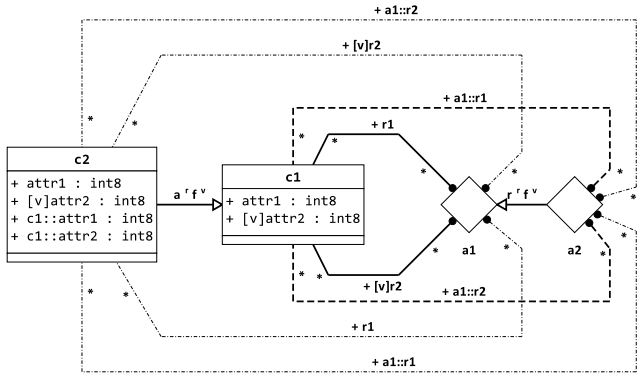


Fig. 17. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating real mode inheritance in the AODB Metamodel, showing the components inherited

of their ability to move: *Flying* and *Swimming*. They virtually inherit the attribute from the *Animal* collection.

$$\Box\{Flying^{\varnothing}, Swimming^{\varnothing}\} \xrightarrow{a^{v}} \Box\, Animal \tag{22}$$

Next, we defined the *WaterBird* collection ($Coll$), which is in relationship of inheritance in natural mode with the aforementioned abstract collections.

$$\begin{gathered}\Box WaterBird \xrightarrow{a} \Box\,\{Flying^{\varnothing}, Swimming^{\varnothing}\} \\ \wedge \Box\,\{Flying^{\varnothing}, Swimming^{\varnothing}\}\langle +Animal :: velocity^{v} : int8\rangle \vDash \\ \Box WaterBird\langle +Animal :: velocity^{v} : int8\rangle\end{gathered} \tag{23}$$

It has inherited single attribute $+Animal :: velocity^{v}$. The most important thing in real inheritance is, that sometimes we need to *devirtualize* a virtual attribute. To show that, we defined the concrete collection ($Coll$) for *Duck* which inherits in real mode from the collection *WaterBird*. It gives the possibility to define the attribute twice, once for each abstract velocity (Fig. 18).

$$\begin{gathered}\Box Duck \xrightarrow{a^{r}} \Box\, WaterBird \wedge \Box WaterBird\langle +Animal :: velocity^{v} : int8\rangle \vDash \\ \Box Duck \left\langle \begin{matrix} +Bird :: Flying :: Animal :: velocity : int8 \\ +Bird :: Swiming :: Animal :: velocity : int8 \end{matrix} \right\rangle\end{gathered} \tag{24}$$

3.4. Algorithm of Inheritance in the Association-oriented Metamodel

Description of the main part of the inheritance algorithm Figure 19 shows an inheritance algorithm common to all three inheritance modes (*n* – natural, *v* – virtual and *r* – real). It also pertains to inheritance of attributes ($Attr$) in collections ($Coll$) as well as roles in associations ($Assoc$).

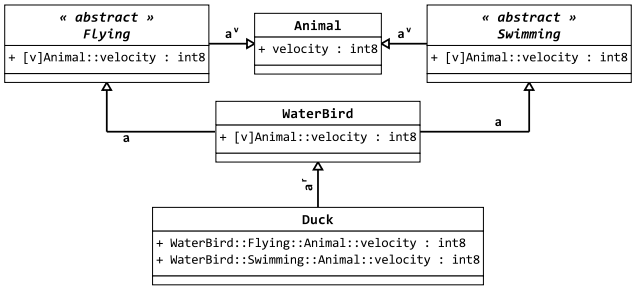


Fig. 18. Diagram developed in the $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ language, illustrating the model in the Example 4

Verification – The verification process is conducted at first in order to check whether one can create a generalisation. It is not possible when:

- a cycle has been found in the graph of generalisation (special case – generalisation of a collection ($Coll$) or an association ($Assoc$) itself to itself),
- incompatibility of databases (Db) has been found for the base and the derived collection ($Coll$) or association ($Assoc$)[2],
- there is already a generalisation established between a specific base and derived element.

If establishing a *gen-spec* relationship has proved impossible, a database system error is generated and the generalisation is not created, whereas the algorithm is finalised.

Creating objects – The inhreitance object is created ($Inheritance$), the generalization base node is set and the object is added to the specialization inheritance list.

Inheritance of components – If component inheritance has been enabled in the course of generalisation, iteration over components of the base collection ($Coll$) or association ($Assoc$) takes place.

- **Inheritability**[3] **aspect** – Depending on the inheritance mode in the aspect of inheritability, inheritability status of a newly created component is:
  - inheritable: base component is inheritable, and the inheritability mode is *nochange*,
  - not inheritable: base component is not inheritable or the inheritability mode is *disable*.

  No base components of not inheritable status are inherited. The foregoing means that they do not appear in the derived component at all[4].
- **Virtuality aspect** – Depending on the inheritance mode in the aspect of virtuality, the inheritance process proceeds in a slightly different manner.

  When the inheritance is established in:
  - **virtual mode** – a virtual component is created,

[2] Theoretically, it would be possible to make an attempt of introducing a *gen-spec* relationship between elements of two different databases (Db). However, such an action is impermissible in the AODB Metamodel.

[3] Inheritability is a concept of AODB Metamodel which determines whether components are inherited or not.

[4] In the *C++* language, private elements are copied to the derived component; they occupy memory but cannot be accessed directly.

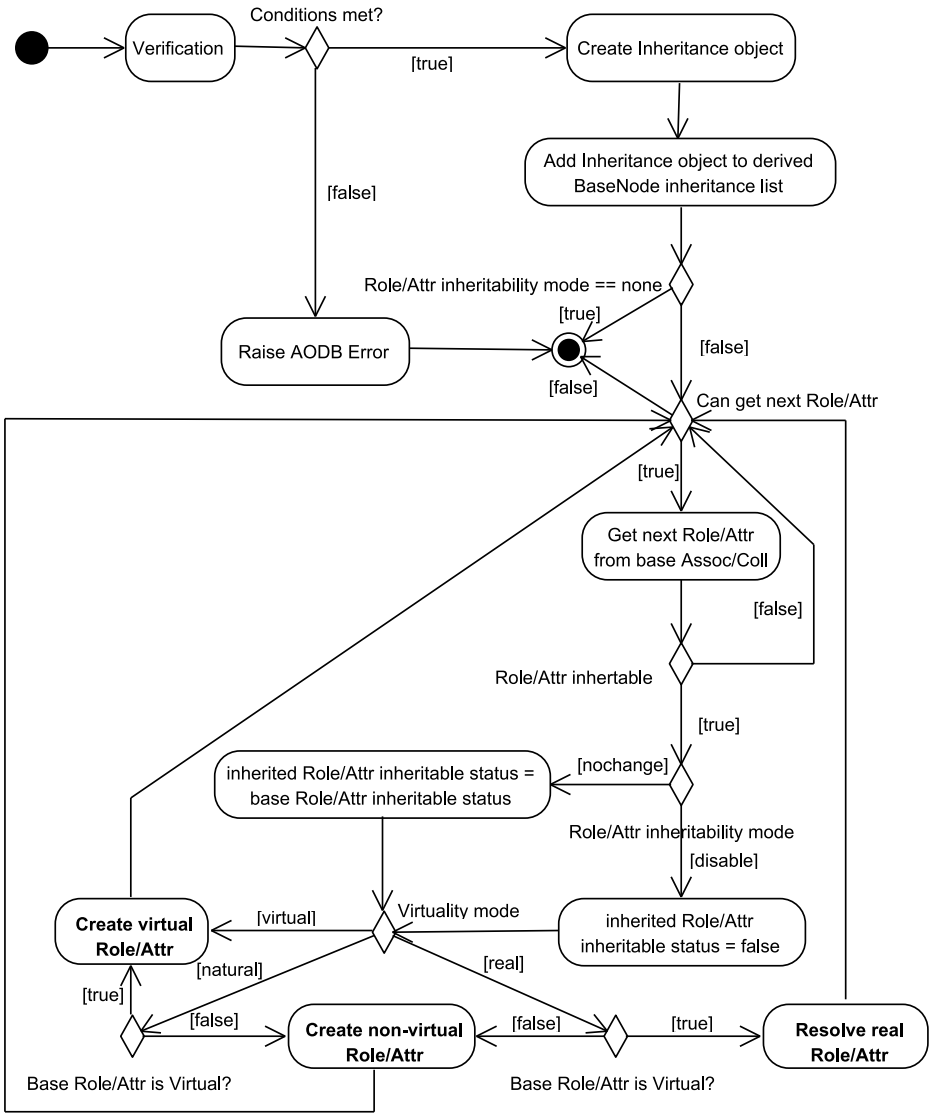


Fig. 19. Activity diagram depicting the component inheritance algorithm in the AODB Metamodel

- **natural mode** – it is verified whether or not the base component is virtual. If it is:
  - **virtual** – a virtual component is created,
  - **non-virtual** – a non-virtual component is created,
- **real mode** – it is verified whether or not the base component is virtual. If it is:

* **virtual** – procedure , as depicted in Figure 20, used to resolve the ambiguity of inherited components, is executed, and this is where they are made "real",
* **non-virtual** – a non-virtual component is created.

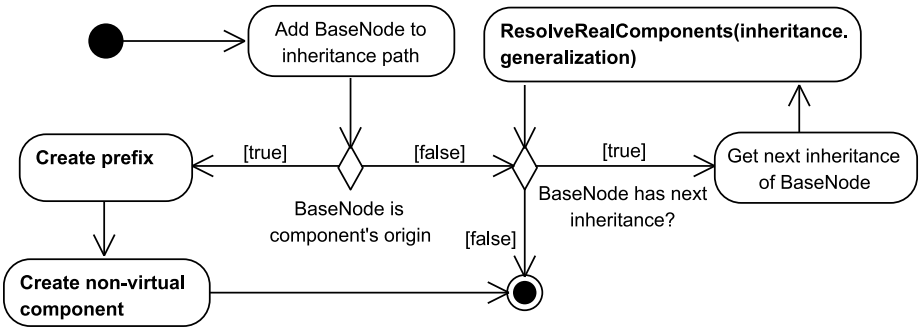


Fig. 20. Activity diagram depicting an algorithm for resolving ambiguity of inherited components –

The algorithm provided in Figure 20 pertains to resolving ambiguity of inherited components – . The following are descriptions of its most important elements:

– The method envisaged to resolve the ambiguity consists is determining the path of virtual inheritance of an attribute ($Attr$) or a role ($Role$) starting from the moment of its creation.
– In order to determine a unique path, consecutive base nodes constituting generalisations of all levels of the pre-set base collection ($Coll$) or association ($Assoc$) are recursively checked. A prerequisite for positive finalisation of a path of function calls is finding a base node being the source of creation of the given virtual component.
– In the course of successive procedure calls, consecutive base nodes, for which the procedure is initiated each time, are added to the path.
– Once the path is determined (along with the source), one has obtained the information required to create a "devirtualised" component. A full path will be needed to create a complete prefix containing all levels of generalisation.

Principles of defining prefixes The inheritance mechanism imposes a necessity to develop principles for building names of components in order to ensure that they are unique and, consequently, that one can make references to them. This problem will be a subject of another paper, describing the manner in which prefixes are designated as well as providing a comparative analysis conducted with reference to an analogical mechanism functioning under an object-oriented programming language, namely *C++*. Below is an overall outline of the relevant principles collated as a list of key items.

- Own components[5] only require their names to be specified.
- Inherited components require that a complete path in the hierarchy of inheritance should be specified[6].
- Prefixes are built of component names, starting from an element of the lowest rank in the inheritance hierarchy[7].
- When the virtual mode is encountered:
  - symbol is entered before the component name,
  - if there is a virtual mode in the inheritance path, and at the same time there is no real mode, the prefix building ends with the first virtual mode encountered.
- When the real mode occurs in the inheritance path, it is from that point forth that all components lose the property of virtuality[8].

4. Applications in Knowledge-Based Systems

Following section is an evaluation of AODB Metamodel in the form of exemplary knowledge representations system structures. Those structures use both collections' and associations' inheritance mechanisms.

In the Ontological Core Module (OCM^{SKB}) structure of SKB – Semantic Knowledge Base (Fig. 21) [24] the mechanism of collection inheritance in the aspects of attributes and rights to fulfill roles was used to define specializations of abstract collection CONCEPT$^{\varnothing}$, e.g. CLASS, INSTANCE, SET, RELATIONSHIP, FEATURE, VALUE, UNIT, VALUESPEC. Moreover, the inheritance of associations was used for the definition of UsedIn association, which is a specialization of Connection association enriched with a Relationship role. Another example might be a SetInstance association being a specialization of both ClassInstance and SetConcept.

SKB Extended Semantic Network Module (ESNM^{SKB}) [28] has been presented on the AML$^{AO}_{DB}$ diagram (Fig. 22). It is designed for the storage of complex rules and facts in the form of hypergraphs. Since this module is oriented mainly on building relationships, thay have their emanation among others in the form of complex associations taxonomy. The central association is Operand$^{\varnothing}$, which is a generalization for following types of operands: OperandOperator, OperandInstance, OperandNode, OperandNet. Additionaly OperandInstance inherits from Instance$^{\varnothing}$, while OperandNode inherits from Substitutability.

$$\Diamond OperandNode \xrightarrow{r} \Diamond\, Substitutability \tag{25}$$

It should be noted that only roles are inherited with no rights to fulfill roles. Moreover, following inheritances occur:

$$\Diamond Operator \xrightarrow{r,f^v} \Diamond\, Instance; \Diamond Conclusion \xrightarrow{r,f^v} \Diamond\, Net. \tag{26}$$

[5] Uninherited.

[6] Exceptions to this rule have been described in the item concerning virtual inheritance.

[7] Component names are separated with a double colon symbol – .

[8] It means that a complete path of all components must be placed in the prefix.

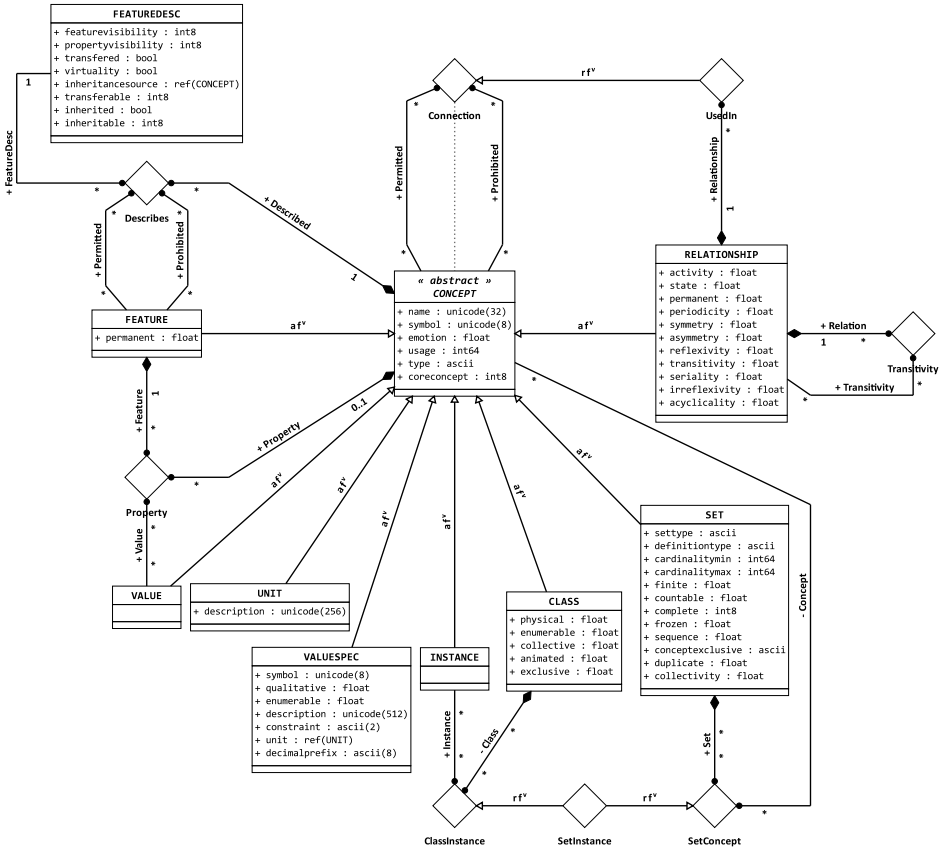


Fig. 21. $\mathsf{AML}^{\mathsf{AO}}_{\mathsf{DB}}$ diagram of $\mathsf{OCM}^{\mathsf{SKB}}$

5. Related Work

The issue of inheritance in database world should be identified as two different approaches: mechanisms directly implementing the semantics of inheritance and mapping of structures according to appropriate design patterns. One of well described standards of direct implementation of inheritance mechanism is OM ODMG 3.0 [6]. In the field of conceptual implementation of inheritance is mapping in O/R [20] or EER [36, 38].

The literature is rich in the discussion on identity of the inheritance mechanism and subtyping mechanism conceptions. The opinions seem to be divided [9, 10]. OM ODMG 3.0 [6] separates and implements both of those concepts. The Generalization-Specialization is emanation of subtyping mechanisms, while Extends ensures inheritance. RDF being a triplestore database metamodel type which provides subtyping mechanism only. The AODB approach towards inheritance assumes coexistence of both aforementioned concepts within one mechanism.

The inheritance mechanism can have semantic character (built from existing syntax elements) or can have predefined syntax and semantics with carefully described gen-

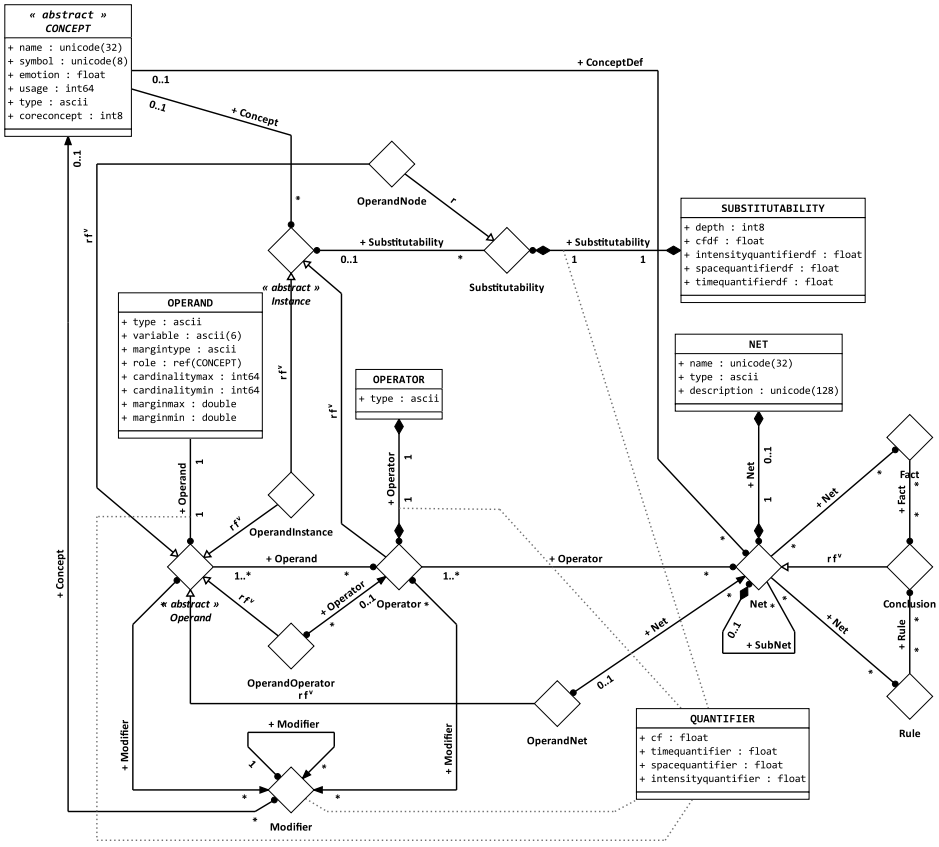


Fig. 22. $\mathrm{AML}^{\mathrm{AO}}_{\mathrm{DB}}$ Diagram of $\mathrm{ESNM}^{\mathbf{SKB}}$

spec mechanism. The inheritance mechanism in ODMG OM 3.0 cover both inheritance of elements state and behavior [37] as well as substitutability in OQL [11]. The inheritance mechanism in EER has been defined taking into account total/partial and disjoint/overlapping properties. While graph databases in most cases represent the *IS_A* relation as an edge named as such, it has nothing in common with inheritance mechanism and it functions only in semantics layer. Ohira et al. [30] proposed the *Directed Recursive Hypergraph data Model*, that seems to fill the gap with the definition of Shape graph providing information in regard of i.a. gen-spec relationships.The RDF metamodel has defined in standard [3, 29] property, that defines all instances of a given subclass being at the same instance of a base class. The
property works analogically for properties. SPARQL is a language using those mechanisms [41]. Relational metamodel [31] has no mechanism of inheritance defined, however there are number of studies providing design patterns and mappings implementing gen-spec relationships [4, 34] that are used e.g. in O/R systems [8]. Some of the high-level approaches to metamodeling, e.g. Context-Driven Meta-Modeling show, that inheritance

issues are not needed in the high level of modeling, but they can be defined in the domain specification language level [39, 40].

The AODB provides inheritance separate mechanisms for associations and datasets. Those mechanisms provide number of modes allowing to selectively inherit components or rights to fulfill roles, as well as they fully support virtual inheritance. Moreover, components have the property named Inheritability, that allow to selectively define which components should be subject to inheritance mechanism.

6. Conclusions

The article addresses the subject of inheritance in the Association-Oriented Database (AODB) Metamodel compared with an analogical solution typical of the object-oriented model. Due to the fact that the AODB Metamodel features a number of fundamental innovations compared to the *object-oriented model*, the most important components of this model are discussed in the first part of the paper. What follows is a description of the AODB inheritance mechanism, analysed from the perspective of specificity of its definition.

The inheritance mechanism is one of the most important and popular methods used to describe complexity. It ensures considerable simplification of the modelling activity. In AODB, inheritance is particularly crucial, and it is strongly integrated with other mechanisms of this model. The foregoing results from the fact that competences of data containers and relationships between data have been separated. This separation is decisive in its influence on all aspects of a model, including the inheritance mechanism. In the intensional[9] sense, the elements responsible for data storage in the object-oriented model, namely attributes and classes, also perform functions enabling relationships between data to be established. Inheritance applies to classes and is responsible for attributes, irrespective (in an indistinguishable manner) of whether the latter are containers for data or constitute relationships between them. On the other hand, in the AODB Metamodel, inheritance independently pertains to data containers, the function being performed by collections ($Coll$), and relationships between data, namely associations ($Assoc$). Attributes ($Attr$) are inherited within a collection ($Coll$), as roles ($Role$) are within an association ($Assoc$). Since both collections ($Coll$) and associations ($Assoc$) can participate in relationships, then also inheritance of rights to fulfil roles is possible within each of these categories. In collections ($Coll$), inheritance of attributes ($Attr$) and rights to fulfil roles is entirely and mutually independent. The situation is analogical in associations ($Assoc$), i.e. inheritance of roles ($Role$) and rights to fulfil roles is also completely separated on the modelling level.

The examples presented here show inheritance methods that significantly support the modeling of relatively complex conceptual constructs. In particular, it should be emphasized that the inheritance within associations allowed for a considerable simplification of the schema, and in many cases made it possible to express concepts that would be impossible to implement in e.g. *C++* or generally speaking *object-oriented model*. It is worth to mention that in AODB it is possible to define inheritance modes, thus separating the aspect of inheriting components from the aspect of the inheritance of roles. The pre-

[9] Structural.

sented examples clearly point out both the significant semantic capacity[10] and the power of expression[11] of AODB Metamodel. Its implementation is an undeniable asset that, combined with the above features, makes it a very powerful tool for modeling complex knowledge representation structures.

Owing to the algorithm developed for prefixing names of the components being inherited, also entailing virtual inheritance modes, a designer is unconstrained in naming components, without any concerns pertaining to ambiguity related to uniqueness of names. At the same time, multiple inheritance is permissible, since by no means does it cause any ambiguity or any limitations as to names of components or inheritance structures. The only limitation, structurally imposed upon inheritance, is absence of cycles, which is detected and controlled by an association-oriented database using internal algorithms ensuring coherence detection as well as correctness of structures and data. The inheritance mechanism functioning in the AODB Metamodel is of major importance for modeling productivity, and its complete integration with other mechanisms exerts very significant influence on the model's power of expression. The proposed solution could make a benefit for modeling ontologies [12] and other knowledge representation systems, as shown on the case studies of OCM^{SKB} and ESNM^{SKB}.

References

1. Alagíc, S.: Type-checking OQL queries in the ODMG type systems. ACM Transactions on Database Systems 24(3), 319–360 (sep 1999)
2. Alashqur, A.M., Su, S.Y.W., Lam, H.: OQL: a query language for manipulating object-oriented databases. In: VLDB '89 Proceedings of the 15th international conference on Very large data bases, pp. 433–442. VLDB '89, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
3. Antoniou, G., van Harmelen, F.: A semantic web primer (2004)
4. Arafi, F.: Supporting inheritance in relational database systems. Proceedings Fourth International Conference on Software Engineering and Knowledge Engineering pp. 511–518 (1992)
5. Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S.: The Object-Oriented Database System Manifesto. In: Proceedings of the First International Conference on Deductive and Object-Oriented Databases, pp. 223–240. North-Holland, Kyoto, Japan (1989),
6. Berler, M., Eastman, J., Jordan, D., Russell, C.L., Schadow, O., Stanienda, T., Velez, F.: The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publishers (2000)
7. Blakeley, J.A.: OQL[C++]: extending C++ with an object query capability. In: Kim, W. (ed.) Modern database systems, chap. OQL[C++]:, pp. 69–88. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1995)
8. Cabibbo, L., Carosi, A.: Managing Inheritance Hierarchies in Object/Relational Mapping Tools. In: 17th International Conference on Advanced Information Systems Engineering, CAiSE 2005, Porto, Portugal. vol. 3520, pp. 135–150 (2005)
9. Cartwright, R., Moez, A.A.: Inheritance is subtyping. In: The 25th Nordic Workshop on Programming Theory (NWPT). Citeseer (2013)

[10] The term **semantic capacity** is used as an identifier of a concept describing potential of a metamodel to express complex conceptual structures.

[11] The term **power of expression** is used as an identifier of a concept describing high semantic content in a concise syntactic form.

10. Cook, W.R., Hill, W., Canning, P.S.: Inheritance is not subtyping. Computing pp. 125–135 (1990),
11. Dhande, S.S., Bamnote, G.: Query Optimization in OODBMS: Identifying Subquery for Complex Query Management. Computer Science & Information Technology (CS & IT) (April), 161–177 (2014)
12. Dudycz, H., Korczak, J.: Process of ontology design for business intelligence system. In: Information Technology for Management - Federated Conference on Computer Science and Information Systems, ISM 2015 and AITM 2015, Lodz, Poland, September 2015, Revised Selected Papers. pp. 17–28 (2015)
13. Garvey, M., Jackson, M.: Introduction to object-oriented databases. Information and Software Technology 31(10), 521–528 (dec 1989)
14. Gray, P.M.D.: Oql. In: LIU, L., M. Tamer Özsu (eds.) Encyclopedia of Database Systems, pp. 2003–2004. Springer US (2009)
15. Habela, P., Roantree, M., Subieta, K.: Flattening the Metamodel for Object Databases. In: Proceedings of the 6th East European Conference on Advances in Databases and Information Systems. pp. 263–276. ADBIS '02, Springer-Verlag, London, UK, UK (2002)
16. Henrich, A.: P-OQL: n OQL-oriented query language for PCTE. In: SEE '95 Proceedings of the 1995 Software Engineering Environment Conferences, pp. 48–60. SEE '95, IEEE Computer Society, Washington, DC, USA (1995)
17. Jodłowiec, M., Krótkiewicz, M.: Information Technologies in Medicine: 5th International Conference, ITIB 2016 Kamień Śląski, Poland, June 20 - 22, 2016 Proceedings, Volume 1. chap. Semantics Discovering in Relational Databases by Pattern-Based Mapping to Association-Oriented Metamodel – a Biomedical Case Study, pp. 475–487. Springer International Publishing, Cham (2016)
18. Jodłowski, A., Habela, P., Płodzień, J., Subieta, K.: Objects and Roles in the Stack-Based Approach. In: Hameurlain, A., Cicchetti, R., Traunmüller, R. (eds.) Database and Expert Systems Applications, Lecture Notes in Computer Science, vol. 2453, pp. 514–523. Springer Berlin Heidelberg (2002)
19. Josifovski, V., Su, S.Y.W.: Incorporating association pattern and operation specification in odmg's oql. In: Proceedings of the Sixth International Conference on Information and Knowledge Management. pp. 332–340. CIKM '97, ACM, New York, NY, USA (1997)
20. Keith, M., Schnicariol, M.: Object-Relational Mapping. In: Pro JPA 2, pp. 69–106 (2009)
21. Kim, W.: Object-oriented databases: definition and research directions. IEEE Transactions on Knowledge and Data Engineering 2(3), 327–341 (1990)
22. Kozankiewicz, H., Stencel, K., Subieta, K.: Distributed Query Optimization in the Stack-Based Approach. In: Yang, L.T., Rana, O.F., Di Martino, B., Dongarra, J. (eds.) High Performance Computing and Communications, Lecture Notes in Computer Science, vol. 3726, pp. 904–909. Springer Berlin Heidelberg (2005)
23. Krótkiewicz, M.: Association-Oriented Database Model – n-ary Associations. International Journal of Software Engineering and Knowledge Engineering 27(02), 281–320 (mar 2017)
24. Krótkiewicz, M., Jodłowiec, M., Wojtkiewicz, K.: Introduction to Semantic Knowledge Base : multilanguage support of Linguistic Module. In: 2016 Third European Network Intelligence Conference (ENIC 2016). IEEE Computer Society Conference Publishing Services, Wrocław (2016)
25. Krótkiewicz, M., Wojtkiewicz, K.: Conceptual Ontological Object Knowledge Base and Language. In: Computer Recognition Systems, vol. II, pp. 227–234. Springer (2005)
26. Krótkiewicz, M., Wojtkiewicz, K.: Introduction to semantic knowledge base: Linguistic module. In: 2013 6th International Conference on Human System Interactions (HSI). pp. 356–362. IEEE, Sopot, Poland (jun 2013)
27. Krótkiewicz, M., Wojtkiewicz, K.: Functional and Structural Integration without Competence Overstepping in Structured Semantic Knowledge Base System. Journal of Logic, Language and Information 23(3), 331–345 (sep 2014)

28. Krótkiewicz, M., Wojtkiewicz, K., Jodłowiec, M., Pokuta, W.: Semantic knowledge base: Quantifiers and multiplicity in extended semantic networks module. In: Communications in Computer and Information Science. vol. 649, pp. 173–187. Springer, Cham (2016)
29. McBride, B.: Rdf vocabulary description language 1.0: Rdf schema (2004),
30. Ohira, Y., Hochin, T., Nomiya, H.: Introducing Specialization and Generalization to a Graph-Based Data Model, pp. 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
31. Paredaens, J., De Bra, P., Gyssens, M., Van Gucht, D.: The structure of the relational database model, vol. 17. Springer Science & Business Media (2012)
32. Płodzień, J., Kraken, A.: Object Query Optimization in the Stack-Based Approach. In: Eder, J., Rozman, I., Welzer, T. (eds.) Advances in Databases and Information Systems, Lecture Notes in Computer Science, vol. 1691, pp. 304–316. Springer Berlin Heidelberg (1999)
33. Snyder, A.: Encapsulation and inheritance in object-oriented programming languages. SIGPLAN Not. 21(11), 38–45 (Jun 1986)
34. Stathopoulou, E., Vassiliadis, P.: Design Patterns for Relational Databases (i), 1–38 (2009)
35. Subieta, K., Beeri, C., Matthes, F., Schmidt, J.W.: A Stack-Based Approach to Query Languages. In: Eder, J., Kalinichenko, L. (eds.) East/West Database Workshop, pp. 159–180. Workshops in Computing, Springer London (1995)
36. Teorey, T.J., Yang, D., Fry, J.P.: A logical design methodology for relational databases using the extended entity-relationship model. ACM Comput. Surv. 18(2), 197–222 (Jun 1986)
37. Torres, M., Samos, J.: Generation of external schemas in odmg databases. In: Database Engineering and Applications, 2001 International Symposium on. pp. 89–98. IEEE (2001)
38. Upadhyaya, S.R., Kumar, P.S.: Eronto: A tool for extracting ontologies from extended e/r diagrams. In: Proceedings of the 2005 ACM Symposium on Applied Computing. pp. 666–670. SAC '05, ACM, New York, NY, USA (2005)
39. Zabawa, P.: Context-Driven Meta-Modeling Framework (CDMM-F) – Internal Structure. e-Informatica Software Engineering Journal (2017), accepted for publication
40. Zabawa, P., Hnatkowska, B.: CDMM-F – Domain Languages Framework. In: Świątek, J., Borzemski, L., Wilimowska, Z. (eds.) Information Systems Architecture and Technology: Proceedings of 38th International Conference on Information Systems Architecture and Technology – ISAT 2017: Part II. pp. 263–273. Springer International Publishing, Cham (2017)
41. Zhai, J., Zhou, K.: Semantic retrieval for sports information based on ontology and SPARQL. Proceedings - 2010 International Conference of Information Science and Management Engineering, ISME 2010 1(4), 395–398 (2010)

A. Elements of formal notation of the AODB

While defining the AODB Metamodel, the formal notation described below was envisaged.
$\{\tau\}$ – set of type τ elements. $[\tau]$ – table of type τ elements. $\langle\tau\rangle$ – list of type τ elements. $(t_1 : \tau_1, t_2 : \tau_2, \ldots, t_n : \tau_n)$ – tuple containing elements $t_1, t_2, \ldots, t_n$ of types $\tau_1, \tau_2, \ldots, \tau_n$ respectively.
$Coll :: c_1$ or $\Box c_1$ – collection ($Coll$) named c_1. $Assoc : a_1$ or $\Diamond a_1$ – association ($Assoc$) named a_1. $Obj :: o_1$ or $\boxdot o_1$ – object (Obj) named o_1.
$AssocObj :: ao_1$ or $\diamondsuit ao_1$ – association object ($AssocObj$) named ao_1. $o_1 : c_1$ or $\boxdot o_1 : \Box c_1$ – object (Obj) o_1 of collection ($Coll$) c_1. $o_1 \in c_1$ – object (Obj) o_1 extensionally belongs to collection ($Coll$) c_1, in the case of association objects ($AssocObj$) $ao_1 \in a_1$.
$c_1 \mapsto objs[i]$ – reference to the i^{th} object of collection ($Coll$) named c_1.

$c_1\,\langle +at_1 : int8, +at_2^v : int8\rangle$ – attributes ($Attr$) of collection ($Coll$).
$c_1 \mapsto attrs$ – reference to element $attrs$ of a tuple of collection ($Coll$) named c_1, which means a reference to a list of attributes ($Attr$) of collection ($Coll$) marked as c_1.
$o_1.at_1$ – reference to attribute ($Attr$) at_1 of object (Obj) o_1.
$(v_1 : \tau_1, v_2 : \tau_2, \ldots, v_n : \tau_n)$ – tuple representing values ($Value$) of attributes ($Attr$) of object (Obj), where $v_1, v_2, \ldots, v_n$ are type ($Value$) values.
$\Box c_1^{\varnothing}$ – abstract collection ($Coll$). $\Diamond a_1^{\varnothing}$ – abstract association ($Assoc$). $\Box c_1^{\times}$ – non-navigable collection ($Coll$). $\Diamond a_1^{\times}$ – non-navigable association ($Assoc$).
States of inheritability of components, i.e. attributes ($Attr$) and roles ($Role$): + – *inheritable*, – – *noninheritable*.
Role: $\overset{+r}{\bullet\!\!\longleftarrow}$ – $biNav, biDir$, $\overset{+r}{\bullet\!\!\longrightarrow}$ – $biNav, toDest$, $\overset{+r}{\bullet\!\!\twoheadleftarrow}$ – $biNav, toOwner$, $\overset{+r}{\bullet\!\!\twoheadrightarrow}$ – $uniNav, toDest$.

$a_1 \left\langle \begin{array}{c} [1] \overset{+r_1(\Box c_1)}{\bullet\!\!\longleftarrow} [*]\,\Box\, c_1, \\ [0..1] \overset{+r_2(\Diamond a_2)}{\bullet\!\!\longleftarrow} [1..*]\,\Diamond\, a_2 \end{array} \right\rangle$ – list of roles of a specific association.

Virtuality: $\#at_1^v$ – virtual attribute. $\overset{r_1^v}{\bullet\!\!\longleftarrow}$ – virtual role.
Multiplicities: $[0..1]$ – one at the maximum, $[1]$ – exactly one, $[1..*]$ – at least one, $[*]$ – any chosen number, $[n..m]$ – minimum n and maximum m.
A composition may occur on one side of the role – $\Diamond a_1 [*] \overset{+r_1(\Box c_1)}{\bullet\!\!\longleftarrow\!\!\blacklozenge} [*]\,\Box\, c_1$, but it may simultaneously occur on both sides – $\Diamond a_1 [*] \overset{+r_1(\Box c_1)}{\blacklozenge\!\!\longleftarrow\!\!\blacklozenge} [*]\,\Box\, c_1$. $a_1 \cdots c_1$ – describing collections. $\longrightarrow\!\!\triangleright$ – inheritance[12].
For the collection ($Coll$), attribute ($Attr$) inheriting is possible in: $\overset{a}{\longrightarrow\!\!\triangleright}$ – *natural* mode, $\overset{a^v}{\longrightarrow\!\!\triangleright}$ – *virtual* mode, $\overset{a^r}{\longrightarrow\!\!\triangleright}$ – *real* mode, whereas inheriting rights to fulfil roles – only in the virtual mode: $\overset{f^v}{\longrightarrow\!\!\triangleright}$.
For the association ($Assoc$), role ($Role$) inheriting may take place in: $\overset{r}{\longrightarrow\!\!\triangleright}$ – *natural* mode, $\overset{r^v}{\longrightarrow\!\!\triangleright}$ – *virtual* mode, $\overset{r^r}{\longrightarrow\!\!\triangleright}$ – *real* mode, whereas inheriting rights to fulfil roles – only in the virtual mode: $\overset{f^v}{\longrightarrow\!\!\triangleright}$.
Moreover, inheritance may take place in: $\overset{a}{\longrightarrow\!\!\triangleright}$, $\overset{r}{\longrightarrow\!\!\triangleright}$, $\overset{f^v}{\longrightarrow\!\!\triangleright}$ – *nochange* mode, $\overset{a_-}{\longrightarrow\!\!\triangleright}$, $\overset{r_-}{\longrightarrow\!\!\triangleright}$, $\overset{f^v_-}{\longrightarrow\!\!\triangleright}$ – *disable* mode, component symbol (a, r, f) absent – *none* mode e.g.: $\overset{a^v_-}{\longrightarrow\!\!\triangleright}$.
$\diamondsuit ao_1 : \Diamond a_1 \overset{r_1(\Box c_1)}{\bullet\!\!\longleftarrow} \langle \boxdot o_1 : \Box c_1 \rangle$ – in extensional aspect.
It means that association object ($AssocObj$) ao_1 of type a_1 is associated with object (Obj) o_1 of type c_1 by means of role ($Role$) r_1.
$\diamondsuit ao_1 : \Diamond a_1 \overset{r_1(\Box c_1)}{\bullet\!\!\longleftarrow} \langle \boxdot \{o_1, o_2, o_3\} : \Box c_1 \rangle$ – is fully analogical compared to the previous one, the only difference being that the association object ($AssocObj$) is associated with a set of three objects: $\{o_1, o_2, o_3\}$.

$\diamondsuit ao_1 : \Diamond a_1 \left\{ \begin{array}{l} \overset{r_1(\Box c_1)}{\bullet\!\!\longleftarrow} \langle \boxdot o_1 : \Box c_1 \rangle, \\ \overset{r_2(\Box c_2)}{\bullet\!\!\longleftarrow} \langle \boxdot o_2 : \Box c_2 \rangle \end{array} \right\}$ – association ($Assoc$) with multiple roles.

$ao_1 \cdots o_1$ – describing objects.

[12] Understood as a mechanism defined in AODB.

Marek Krótkiewicz received his PhD in Computer Science in 2001 at Wrocław University of Science and Technology (WUST). Since then he focused his research on data structures and knowledge representation. In 2004 he founded the Knowledge and Information Engineering Group, that is a developer of Semantic Knowledge Base (SKB) project. SKB is a hybrid knowledge representation system that uses multiple types of KR methods, such as frames, ontologies, semantic networks etc. For the purpose of SKB he has modeled and implemented Association-Oriented Database (AODB) Metamodel, which is the next generation database metamodel that puts relationships in the center of its interest. He is an author or co-author of more than 20 conference papers and journal articles in regard of SKB and AODB. Currently he holds the position of Assistant Professor at Wrocław University of Science and Technology, where he continues his research career as part of prof. Ngoc-Thanh Nguyen knowledge engineering team.

Received: June 30, 2017; Accepted: December 10, 2017.