Formalism Describing the Object Models for Domains on a Unified Metamodel of Object-Oriented Database Applications Terms

Pavel P. Oleynik*

Shakhty Institute (branch) of Platov South Russian State Polytechnic University (NPI), Russia, Rostov-on-Don, Russia

Abstract: This article presents a formal description of object models in terms of unified object-oriented database applications metamodel. The metamodel was developed by the author and independent of the domain area. The metamodel is realized by the author in own environment of development SharpArchitect RAD (Rapid Application Development) Studio. On the basis of the metamodel we using set theory proposed formal approach to the description of the model application domains.

Keywords. Object-Oriented programming, Object-Oriented databases, Object system metamodel, Formal description of object models.

1. INTRODUCTION

Dominant methodology used in developing applications is currently the object-oriented approach. This article presents a formal description of object models applied domains in terms of unified object-oriented database applications metamodel. In this case describes all kinds of metaclasses, which allow to describe all kinds of entities in the object system, as well as the composition and structure of each metaclass. In addition, attention is paid to the constraints imposed on the formal model, and a list of valid values in the case where it is possible. In the formal model of drawbacks of many existing works, a brief description is presented in this article. The metamodel is realized by the author in own environment of development SharpArchitect RAD Studio.

The article describes the current version of the unified metamodel. Will describe main types of metaclasses to represent classes, attributes, validation rules, and visualization, methods and events entities allocated developer with object-oriented design.

2. REVIEW OF EXISTING PUBLICATIONS

The object-oriented paradigm is the most popular approach currently applied in the development of various applications, including database applications. This paradigm appeared for a long time, but in contrast to other approaches (eg. relational data model) has no precise formal description. However, there are many attempts of formalization.

In [1] the authors propose a formal model of language Z as a set of operators representing a notation for object-oriented language. The authors introduce a technique based on model specifications that describe the transformation of the input database schemes to the output. Wherein said many of steps and rules of such a transformation. Set of operators allows us to describe the various operations that can be combined to produce applications. These operators are combined according to the rules set theory and the theory of the predicate calculus, which makes the resulting approach is intuitive and easy to use.

In [2] presents an approach which used formalization UML (Unified Modeling Language) class diagrams using the Z notation. Feature of the work is that in addition to the static component (the property values of class instances) paid attention to the description of the behavior of objects. Given a formal model of the dynamic component provided in the object-oriented paradigm in the form of class methods, which reflect a variety of options on lot of resulting values. Also in the article there is a description of the important features, called class inheritance allows simplify the reuse of pre-built components.

In [3] the authors present their own algebra for object-oriented databases is used. Main feature of OODB (Object-Oriented Database) is the identification of objects which based on object identifiers. Because these OIDs (Object Identifier) are unique, then it is possible to determine object which uniquely associated with the identifier. This is the approach used by the

*Address correspondence to this author at the Shakhty Institute (branch) of Platov South Russian State Polytechnic University (NPI), Russia, Rostov-on-Don, Russia; Tel: +7 (908) 507-80-61;
E-mail: xsl@list.ru

E-ISSN: 2410-2938/15

authors, which address a variety of identifiers as a key element of the model. Then the authors extend their algebra of operators on the set theory and classical operators like union, intersection and difference of sets in the form of the operators of the algebra of object identifiers, which resulted in a new set of objects.Other algebraic theory of object-oriented applications is described in [4]. This work is based on the notion of an abstract data type (ADT), consisting of nested objects connected to the outer class using associations. Key concepts are the sorts that allow us to describe the set of all existing objects (instances of classes) and a set of operations that manipulate the elements of sort. Based on these concepts, the authors make a number of definitions, which allow them to be designed as conventional structures such as sets, lists, arrays, and combinations thereof unlimited nesting (*e.g.* an array of sets).

In [5] attempts to provide an object model in the form of theories. Theoretic-based approach proposed by the authors involves description of the various data structures (eg. arrays) in the form of specifications, which have different sections to describe the static (data fields) and behavioral components (operations used over-specified structure). The authors were able to present even multiple inheritance, and some semblance of events, the concept of which corresponded to the time of writing (1995). In this case, attention is paid to the principles of building a graphical user interface and the subsequent transformation of the current user in the operators of library theories.

Another theory-based approach to the formalization of object-oriented models for application domains in [6] is presented. This work significantly extends the results of [5] and introduces a number of additional sections in the description of the ADT (which in terms of the author likes the class programming language). The work paid attention to formalizing description of operations, methods, and events, which acts as the basis the predicate calculus. This approach allowed the authors to present only the most simple operation, but it is enough in most cases. To describe the behavioral component was used a finite state machine with a deterministic set of states described in the class specific section of the specification. Also paid much attention to describes associations, such as the composition and aggregation. This is a significant step forward, because allows to simulate the relationship of the form "part-whole" and manage the lifecycle of embedded objects.

Despite a number of advantages, all the works were written a long time before and devoted to the general description of minimal design of object-oriented paradigm. However, progress is not in place, programming languages evolve, they appear new syntax. For example, currently use event is an integral part of any larger information system. All the above works do not pay enough attention. In addition, validation has become an essential element in any complex system, since helps to avoid errors when the user inputs data and saving into database in a consistent state in accordance with corporate business rules, which can not be implemented on the side of the DBMS. Described work does not pay attention to the creation of a formal description of the validation rules.

For the user, the information system is very important to a convenient location on the form of graphic elements and visual color and highlight important data. Mechanisms are needed to describe the rules of visualization. In the reviewed articles that are not neglected.

The development of applications increasingly being used design patterns [4]. Often in design used pattern Model-View-Controller (MVC), which allows to describe the behavior controllers to control the application. The formalization of these controllers is not considered in the works.

At a present day, the reports are an integral part of many information systems. They are formed by the user on the basis of information available in the system. Description formal model for reporting missing in the described works.

All the missing elements are very important in the modern world application, what why it pays attention to the unified object-oriented applications metamodel whuch used for implementation of object-oriented database applications. Formal model used to describe the application of the program domains described in the following sections. At the same time used publicly available set theory, which is used by the authors to analyze the work before.

3 UNIFIED OBJECT-ORIENTED APPLICATIONS METAMODEL

In this section a brief look at the metamodel used in the unified environment of rapid development of corporate information systems which called SharpArchitect RAD Studio [7]. In [8-14] was presented complete class diagram metamodel and detailed

assignment of classes. Here we consider only the relevant parts for this article. Figure **1** shows a fragment of a unified object system metamodel with the showing of key associations that are important for further discussion.

Consider some of the main hierarchies of metaclasses. Figure **2** is a diagram of the basic metaclasses used to represent different kinds of classes used to describe the entity classes that are present in the domain model.

The root of hierarchy is an abstract metaclass Class, having inherited two: 1) InheritableClass is used to represent the metaclasses that can be inherited, which support inheritance; 2) NotInheritableClass is used to represent metaclasses that can not be inherited. Metaclass Enum allows to describe enums or sets which have values of a simple type.

Abstract base metaclass CustomAttributeClass used to represent the metaclasses that can have the attributes. DomainClass metaclass is used to represent domain classes. Instances of domain allow us to describe persistent entity classes (such as Customers, Products, Sales), which objects (*e.g.* Ivanov, Bread) are stored in the database. To simplify the description will be called instances of the domain just domain classes (if not assumed otherwise).

Abstract ComputationalClass<TBaseClass> metaclass is the base for all calculated metaclasses those classes, copies of which are not stored in the database and are evaluated at runtime (transient). For example, the Turnover-Balance Sheet is not stored directly in the database, and is calculated based on inventory, receipts and expenditures (which are the domain class and represents an instance of DomainClass metaclass).

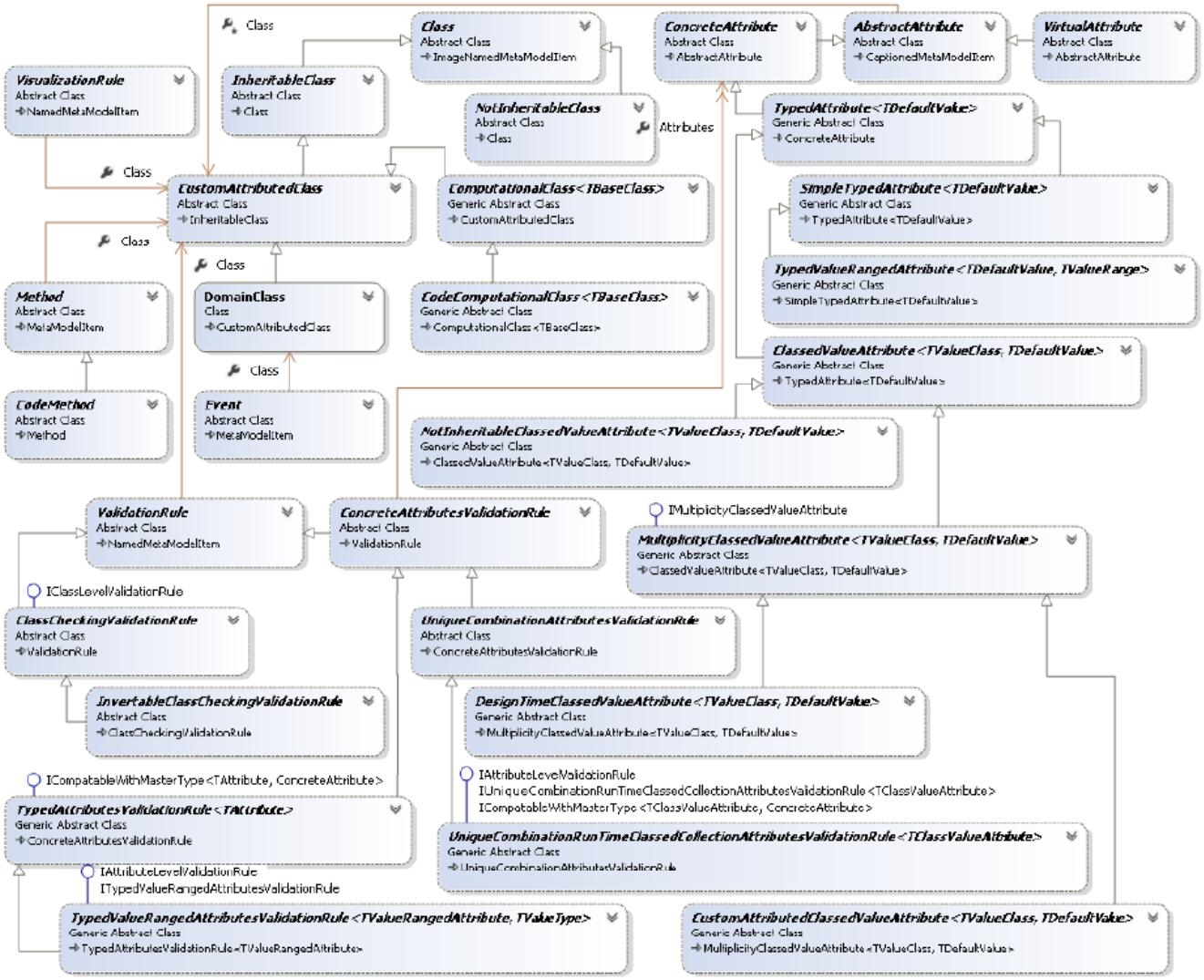

Figure 1: Fragment of a unified object system metamodel.

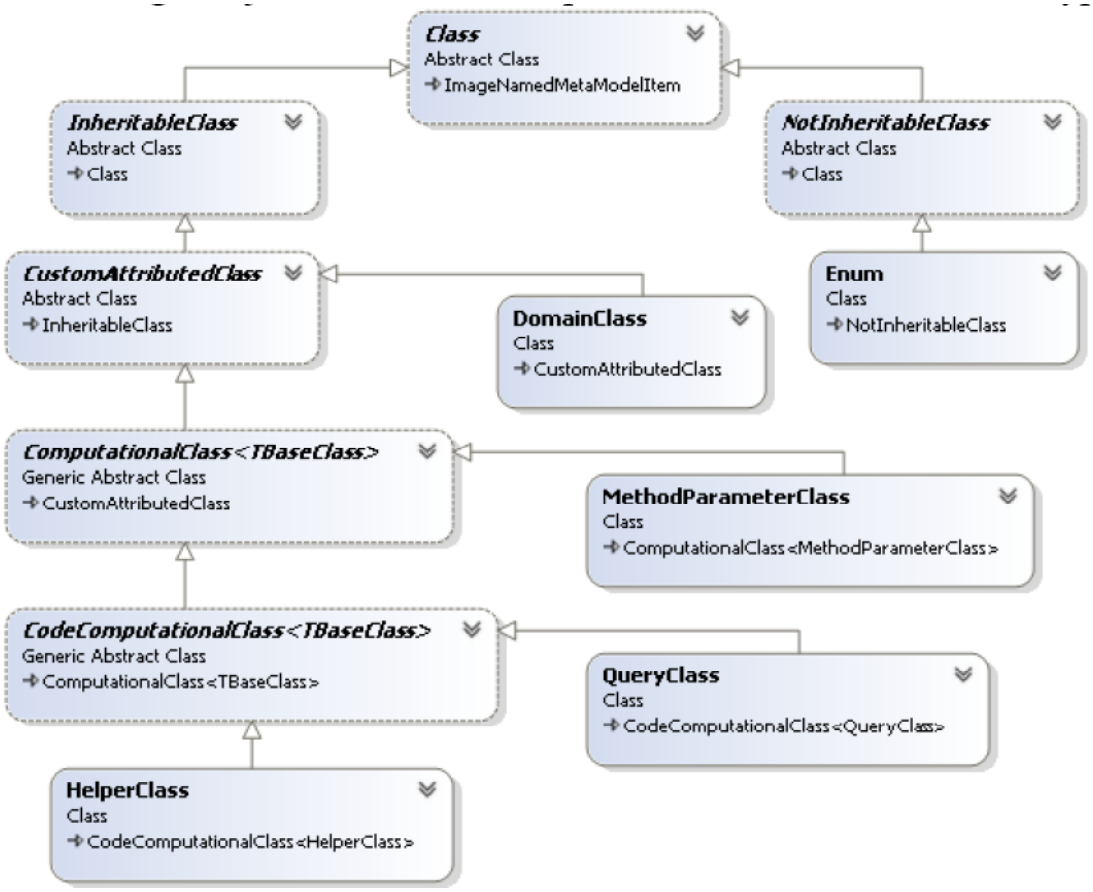

Figure 2: Basic metaclasses used to represent classes of entities from the domain model.

MethodParameterClass metaclass is used represent a class-parameter methods. In SharpArchitect RAD Studio implemented a design pattern which called Parameter object, the essence of which is the transfer of a set parameters in the method as a single object instead of multiple variables with atomic data type.

Abstract CodeComputationalClass<TBase Class> metaclass is the base for calculated metaclasses implemented using software code in C#. QueryClass is metaclass for present query, allowing to form on the basis of the result of database queries (often based on LINQ-qeury, but possible, and direct sending SQL-queries). HelperClass used represent the subsidiary metaclasses that can be displayed in the user interface and used for internal purposes in the implementation of business logic.

We now consider the metaclass used to describe the attributes of the classes and shown in Figure **3**.

Root abstract metaclass represents an attribute is AbstractAttribute. Inherited from the VirtualAttribute classes are used to represent the attributes that were not created by the developer of IS for the application domain, and were presented to the system. They are necessary for an understanding of the metamodel and simplify the software development process. SystemAttribute allows us to describe the attributes that are present in the system and in C# language. GeneratedAttribute class is used to represent attributes that are automatically generated by the system. For example, if you inherit from a base class is automatically added to the tree attribute Node, which allows to get the child nodes and thus form a hierarchical structure.

For presentation attributes whose values can be set by end user, will used an abstract base ConcreteAttribute metaclass. Since the system is implemented in language C#, when we can save the values in the database used by the data types of the language. To describe this moment added parameterized metaclass TypedAttribute<TDefaultValue>. TypeAttribute used to represent a property whose value can preserve the value of the data type of C# language.

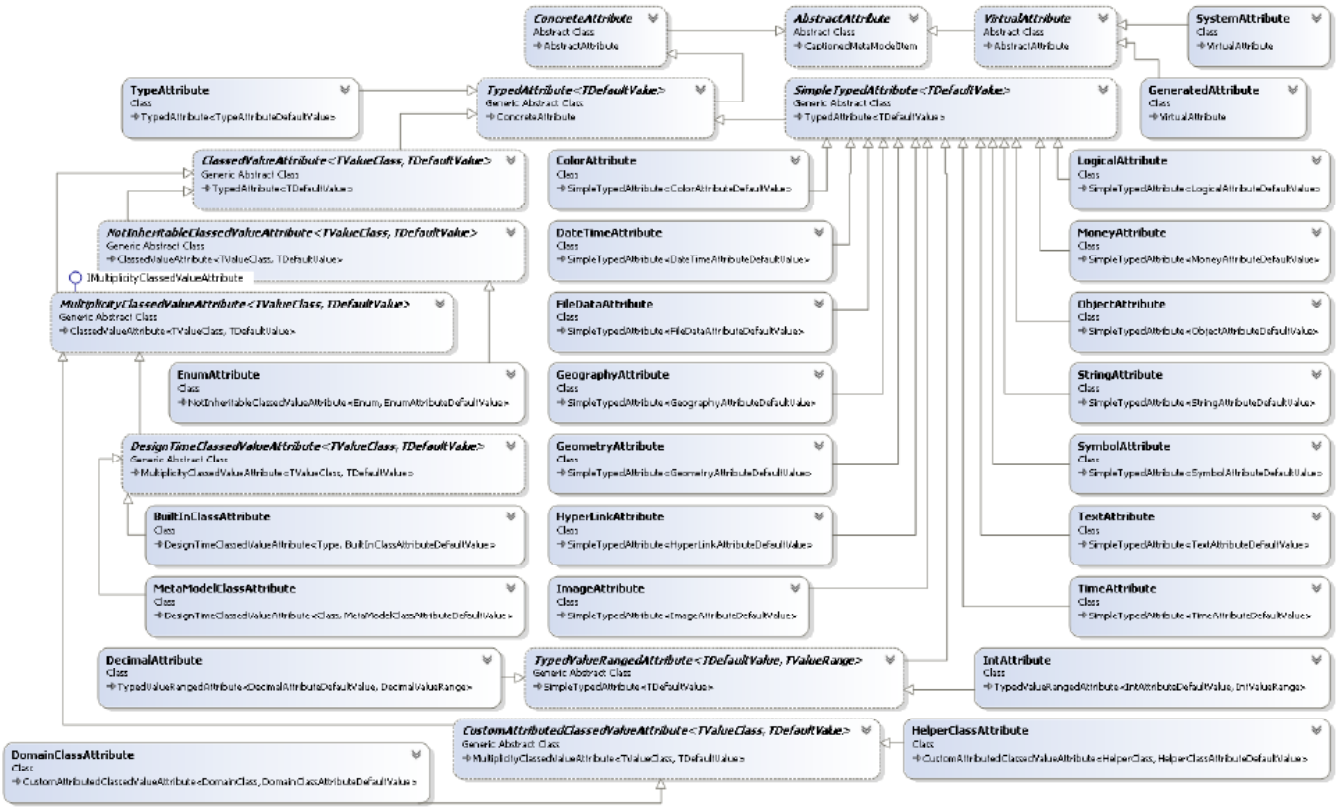

Figure 3: Basic metaclasses which used to describe the attributes of entities.

ClassedValueAttribute<TValueClass, TDefaultValue> metaclass is used to represent attributes whose values are the instances of different classes of entities present in the domain. NotInheritableClassedValueAttribute<TValueClass, TDefaultValue> metaclass keeps copies of non-inherited classes. For example, EnumAttribute, inherited and is used to store enum/set values.

Abstract MultiplicityClassedValueAttribute<TValueClass, TDefaultValue> metaclass is used to represent the values of attributes that can store not only one value (a reference to one instance of the class), but also a collection of values. DesignTimeClassedValueAttribute<TValueClass, TDefaultValue> metaclass allows to save a reference to instances of development time classes. So BuiltInClassAttribute used to store objects implementation classes in the metamodel SharpArchitect RAD Studio. In turn MetaModelClassAttribute allows us to save information about the class metamodel of application domain. Both described metaclasses allow manipulate metamodel at the time of execution of the application. A similar approach is used in many modern programming languages, supports an extensive metainformation. So in C# there is a technology called 'reflection' and allows to impement these things.

CustomAttributedClassedValueAttribute<TValue Class, TDefaultValue> metaclass is used to store instances of classes with attributes. The system has two successor: 1) DomainClassAttribute allows us to save a reference to the instance of an entity subject area described in the metamodel with an instance of a DomainClass metaclass. Attribute of this type is used for the implementation of the association and serves to represent the relations with the object design domain. HelperClassAttribute metaclass allows to store references to instances of HelperClass.

SimpleTypeAttribute<DefaultValue> metaclass is abstract and serves the root of all the attributes which preserving the value of one atomic type (string, number, symbol, etc.). All of this hierarchy is the result of many years of work, the premise of which the intermediate solutions have been described in [15-17]. ColorAttribute metaclass is used to store the color in the format of RGB. LogicalAttribute is used to store the Boolean values (true and false). DateTimeAttribute metaclass is used to save the date-time values. If you want to save only time you should use TimeAttribute.

For input to the money attribute in the hierarchy has a MoneyAttribute metaclass. FileDataAttribute used to save files in difference formats. To save any type of object is useful attribute of type ObjectAttribute. This approach is similar to the use of type 'object' in C#. Type attributes GeographyAttribute and GeometryAttribute used to save the geographical coordinates and geometric objects, respectively. To represent character strings and individual characters are used StringAttribute and SymbolAttribute metaclasses. If you need to save the text of unlimited length with formatting, you must use TextAttribute. HyperLinkAttribute metaclass is used to represent hyperlinks to various resources. ImageAttribute instance is used to store graphics (pictures, photographs and the like). Parameterized abstract TypedValueRangedAttribute<TDefaultValue, TValueRange> metaclass used to represent the atomic values that can be found only in a certain range of values specified by the relevant listing (TValueRange parameter). Inherited IntAttribute metaclass can be used to store integer values, and DecimalAttribute to represent real values.

As can seen SharpArchitect RAD Studio is a mature software product designed for the development of object-oriented database applications, and provides a unified metamodel enabling to describe both static and dynamic elements of the application. Described the development environment has been tested on a variety of projects, described in [11, 14].

4. FORMAL REPRESENTATION OF OBJECT MODELS IN TERMS OF UNIFIED OBJECT-ORIENTED DATABASE APPLICATIONS METAMODEL

To describe a mathematical model allows us to represent the object model of information system any application domain, we use the formal apparatus of set theory. In result, all operations of application can be modeled as of sets manipulations.

The design of information systems necessary build a domain model, containing difference types of entities. For example, the essence of the product, income and expenditure are maintains the essence and the essence of the trial balance is calculated, ie, its instances are not stored in the database directly but calculated on the basis of other entities stored. From a formal point of view, a formal model of the domain model, constructed in terms of a unified object-oriented database application metamodel is a tuple consisting of the sets and presented as a (1):

$$DMFM = (DC, HC, QC, MPC, EC) \quad (1)$$

where:

DC (DomainClass) – a non-empty set of domain classes,

$dmfm \in DMFM, DC \neq \varnothing$;

HC (HelperClass) – set of helper classes;

QC (QueryClass) – set of queries classes;

MPC (MethodParameterClass) – set of method parameters classes;

EC (EnumClass) – set of enumerations and sets of elements classes;

Consider the description of each listed set. Domain classes (DC) are a key set which used for modeling entities of domain which are stored in the database. In general, for a description of all classes of the domain (DC) uses many elements of which is described as a tuple, and (2):

$$DC = \{(ATT, BC_{dc}, M, E, VLR, VSR, BHC, R)\} \quad (2)$$

where the elements of each tuple are defined as follows:

ATT (Attribute) – set of domain class attributes, $ATT \neq \varnothing$ V $BC \neq \varnothing$;

BC_{dc} (BaseClass) – set of base domain classes from which this is derived, $DC \notin BC_{dc}$;

M (Method) – set of class methods that allow to implement the behavior of instances of classes, ie dynamic component;

E (Event) – set of event handlers that occur in the life cycle of the domain class object;

VLR (ValidationRule) – set of predicates which representing the validation rules to be fit by each object;

VSR (VisualizationRule) – set of visualization rules that management visibility, enability and color of each attributes;

BHC (BehaviorController) – set of behavior controllers that management the behavior of objects as well as application user interface;

R (Report) – set of reports which allows show all the instances of a class (object) in user-friendly form with the ability to print data.

It performs the following restriction:

$dc \in DC, \forall bc_{dc} \in BC_{dc} \subsetneq DC=) dc \notin BC_{dc}$

To describe the properties of instances classes must describe the set of attributes ATT, is formally described as a tuple of the form (3):

$$ATT = \{(Name, AttributeKind, Multiplicity, dc, hc, ec)\} \quad (3)$$

where:

Name – unique name of the attribute (the correct ID);

AttributeKind – kind of attribute;

Multiplicity – multiple attributes (minimum and maximum number of associated objects);

dc – domain class ($dc \in DC$), indicates when AttributeKind = DomainClassAttribute или AttributeKind = GeneratedAttribute;

hc – helper class ($hc \in HC$), indicates when AttributeKind = HelperClassAttribute или AttributeKind = GeneratedAttribute;

ec – enum / set class ($ec \in EC$), indicates when AttributeKind = EnumAttribute.

The system provides the developer with many different types of attributes that can be represented as (4):

AttributeKind = {BuiltInClassAttribute, ColorAttribute, DateTimeAttribute, DecimalAttribute, DomainClassAttribute, EnumAttribute, FileDataAttribute, GeographyAttribute, GeometryAttribute, HelperClassAttribute, HyperLinkAttribute, ImageAttribute, IntAttribute, LogicalAttribute, MetaModelClassAttribute, MoneyAttribute, ObjectAttribute, StringAttribute, SymbolAttribute, TextAttribute, TimeAttribute, TypeAttribute} (4)

where:

BuiltInClassAttribute – Built-class attribute is used to save the instance of the metamodel

ColorAttribute – The color attribute is used to represent an integer constant describing color

DateTimeAttribute – Date-time attribute is used to represent the date and time

DecimalAttribute – Decimal attribute whose value is a decimal number

DomainClassAttribute – Domain class attribute whose value is an instance (object) of the domain class. Is often used to present associations between classes

EnumAttribute – Enum/Set attributes used to store enumeration and set values

FileDataAttribute – File attribute is used to store the contents of the file

GeneratedAttribute – Generated attribute is used to represent attributes that are automatically generated by the system

GeographyAttribute – Geographic attribute is used to represent geographic coordinates

GeometryAttribute – Geometric attribute is used to store geometric objects

HelperClassAttribute – Helper class attribute is used to store an instance of the helper classes to implement computing

HyperLinkAttribute – Hyperlinked attribute whose value is a hyperlink

ImageAttribute - Graphic attribute is used to store images

IntAttribute – Integer attribute that is used to store integer values

LogicalAttribute – Boolean attribute is used to store a boolean value (0 or 1)

MetaModelClassAttribute – Metamodel class attribute is used to save the instance of the class definition

MoneyAttribute – Money attribute is used to store values in the currency

ObjectAttribute – Object attribute is used to store objects of any type

StringAttribute – String attribute is used to store the strings

SymbolAttribute – Symbol attribute is used to store one character

TextAttribute – Text attribute is used to store the text of unlimited length with formatting

TimeAttribute – Time attribute is used to store the time only

TypeAttribute – Type attribute used to store the name of the data type

Valid values for multiplicity are:

Multiplicity = {0..1, 1..1, 0..*, 1..*}

Since it is assumed bidirectional association implemented using attributes to describe two different ends, something can be described in a similar way other types of multiplicity, for example, a many-to-many (* .. *). Note that the required elements in the expression (4) are only Name, AttributeKind, the name and type of the attribute. The remaining components are not required and may not be available in the description of the attribute.

The set M allows for the behavior of instances of classes in the form of methods, ie dynamic component and can be represented in the form (5):

M = {(Name, mpc, Body)} (5)

where:

Name – The name of the method (valid ID);

mpc – Method parameter, mpc ∈ MPC;

Body – The lines of program code that implements the method.

Each method is a procedure (function does not return a result). When this function may have parameters representing element of MPC. Implemented design pattern called 'Parameter object' when passed as the parameter instance of the class.

The set E is a set of event handlers that occur in the life cycle of the object (instance) essentially describes a class domain and presented in the form of (6):

E = {(Name, EventKind, Body)} (6)

where:

Name – The name of the event handler (the correct ID);

EventKind – Type of event for which the handler is declared;

Body – The lines of program code that implements the event handler.

To describe the types of events used follow enum (7):

EventKind = {AfterChangedAttributeValueEvent, AfterDeletedEvent, AfterLoadedEvent, AfterSavedEvent, BeforeDeletingEvent, BeforeSavingEvent, InitializationEvent} (7)

where:

AfterChangedAttributeValueEvent - An event called after changing the value attribute of an object

AfterDeletedEvent - The event is called after an object is removed

AfterLoadedEvent - An event called after the object is loaded

AfterSavedEvent - Events that trigger after saving the object

BeforeDeletingEvent - Event is called before deleting

BeforeSavingEvent - Event that cause the object before saving

InitializationEvent - Event object when initialization

Set of predicates representing the validation rules to be met by each object is represented by sets VLR, which is described (7):

VLR = {(cr, ATT_{VLR})} (7)

where:

cr – Predicate that has parameters that perform class attributes, which should correspond to instance of an entity domain;

ATT_{VLR} - set of attributes, which used in validation rule.

To improve data analysis and simplify the

presentation uses a variety of visualization rules VSR, is an unordered triples of the form (8):

$$VSR = \{(cr, vrk, ATT_{vsr})\} \quad (8)$$

where:

cr – predicate that determines the applicability of the visualization rules

vrk – determines the type of visualization rules

ATT_{vsr} – a set of attributes that is subject to visualization rule

It performs:

$ATT_{vsr} \subseteq ATT$

$vrk \notin \in \varnothing \ \forall \ VRK$

where:

VRK = {HideProperty, DisableProperty, SetFontColor, SetBackgroundColor}

where:

HideProperty – This type of rule hides attributes

DisableProperty – This type of rule is made inactive attributes

SetFontColor – This type of rule set the color of the font in the attribute editor

SetBackgroundColor - This type of rule sets the background color in the attribute editor

Set of behavior controllers BHC consists of elements each of which is a class of programming language that implements the desired functionality.

Set of reports R is a set of reports, each of which describes an extensible markup language XML and contains data that is interpreted by the application.

We proceed to consider other types of classes that represent the individual elements of the metamodel.

In (9) presents a formal description of a set of helper classes (HC):

$$HC = \{(ATT, bc_{hc}, PC, M, VSR, BHC, R)\} \quad (9)$$

where:

ATT (Attribute) – set of attributes of the domain class;

bc_{dc} (BaseClass) – base helper class from which this is derived;

PC (ProgramCode) – code implementation helper class is represented as a set of rows language C#;

M (Method) – the set of class methods that allow to implement the behavior of instances of classes, ie dynamic behavior;

VSR (VisualizationRule) – set of visualization rules that management visibility and colors of individual attributes;

BHC (BehaviorController) – set of behavior controllers that govern the behavior of objects as well as application user interface;

R (Report) – set of reports which allows to display instances of a class (object) in an easy to user view with the ability to print data.

At the same time, the following restrictions:

$\forall$ umm $\in$ UMM $\forall$ hc $\in$ HC, ATT $\neq \varnothing$ V $bc_{dc} \neq \varnothing$ V PC $\neq \varnothing$

$\forall \varnothing$ hc $\in$ HC, $\forall$ $bc_{hc} \in$ HC => hc $\neq bc_{hc}$

In (10) presented a formal description of a set of queries classes (QC):

$$QC = \{(ATT, PC, VSR, BHC, R)\} \quad (10)$$

where:

ATT (Attribute) – set of attributes of the query class;

PC (ProgramCode) – code implementation of the helper class is represented as a set of rows language C#;

VSR (VisualizationRule) – set of visualization rules that management visibility and colors of individual attributes;

BHC (BehaviorController) – set of behavior controllers that govern the behavior of objects as well as application user interface;

R (Report) – set of reports which allows to display instances of a class (object) in an easy to user view with the ability to print data.

At the same time, the following restrictions:

$\forall$ umm $\in$ UMM $\forall$ qc $\in$ QC, ATT $\neq \varnothing$ V PC $\neq \varnothing$

In (11) presented a formal description of a set of method parameter classes:

$$MPC = \{(ATT, bc_{mpc}, UM, M, VLR, VSR, BHC, R)\} \quad (11)$$

where:

ATT (Attribute) – set of attributes of the method parameter class;

bc_{mpc} (BaseClass)– base method parameter class from which this is derived;

UM (UsingMethod) – set of methods that use this class as a parameter;

M (Method) – set of class methods that allow to implement the behavior of instances of classes, ie dynamic component;

VLR (ValidationRule) – set of predicates representing the validation rules to be fit by each object;

VSR (VisualizationRule) – set of visualization rules that management visibility, enability and color individual attributes;

BHC (BehaviorController) – set of behavior controllers that govern the behavior of objects as well as application user interface;

R (Report) – set of reports which allows to display instances of a class (object) in an easy to user view with the ability to print data.

At the same time, the following restrictions:

$\forall$ umm $\in$ UMM $\forall$ mpc $\in$ MPC, ATT $\neq \varnothing$ V $bc_{mpc} \neq \varnothing$

$\forall$ mpc $\in$ MPC, $\forall$ $bc_{mpc} \in$ MPC =) mpc $\neq bc_{mpc}$

In (12) presented a formal description of a set of enumerations classes(EC), representing the named constants with the assigned integer values:

$$EC = \{(Name, ek, \{val_i = 2^i-1\})\} \quad (12)$$

where:

i $\in$ 0..n;

ek $\in$ EnumKind;

EnumKind = {Enum, Set} – type of enum class, Enum - describes the enumeration and Set - is set.

From these formulas we see that highlighted all the key moments of the object metamodel.

5. CONCLUSION

The result of this paper was developed a formal description of any object models that allow us to apply the approach object-oriented design. In the future to expand the formal apparatus with the development of the metamodel, add a variety of options present in the moment. For example, needs options for specification abstract classes, as well as an option to indicate the need to save copies of classes in the database. The following articles intended to develop a formal description of the constraints imposed on the described model and thereby implement a mechanism for validating models for application domains. You must also complete a formal description of a large test object model. It is supposed to perform it for the model described in [18].

REFERENCES

[1] Periyasamy K, Alagar VS, Subramanian S. Deriving test cases for composite operations in Object-Z specifications. Proc. Technology of OO Languages and Systems (TOOLS 26), Santa Barbara, CA, August 1999, pp. 429-441.

[2] Shroff M, France R. Towards a Formalization of UML Class Structures in Z, Proceedings, 21st International Computer Software and Applications Conference (COMPSAC'97), August 1997, Washington DC, pp. 646-651. http://dx.doi.org/10.1109/cmpsac.1997.625087

[3] Shugang Wang. Object identity set algebra for object-oriented database systems, 5th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2012), 2012, pp. 1-6. http://dx.doi.org/10.1109/SOCA.2012.6449439

[4] Yu XM, Dillon TS. An Algebraic Theory of Object-Oriented Systems. IEEE Transactions on Knowledge and Data Engineering archive, Volume 6 Issue 3, June 1994, pp. 412-419.

[5] DeLoach S, Bailor P, Hartrum T. Representing object models as theories. Proceedings 10th Knowledge-Based Software Engineering Conference, Nov 1995, pp. 28-35. http://dx.doi.org/10.1109/kbse.1995.490116

[6] Scott D. DeLoach, Thomas C. Hartrum. A Theory-Based Representation for Object-Oriented Domain Models. IEEE Transactions on Software Engineering, Volume 26 Issue 6, June 2000, pp. 500-517.

[7] Oleynik PP. Computer program "The Unified Environment of Rapid Development of Corporate Information Systems SharpArchitect RAD Studio", the certificate on the state registration № 2013618212/ 04 september 2013. (In Russian).

[8] Oleynik PP. Class Hierarchy of Object System Metamodel // Object Systems – 2012: Proceedings of the Sixth International Theoretical and Practical Conference. Rostov-on-Don, Russia, 10-12 May 2012. Edited by Pavel P. Oleynik. 37-40 pp. (In Russian), http://objectsystems.ru/files/2012/Object_Systems_2012_Proceedings.pdf

[9] Oleynik PP. Class Hierarchy for Presentation Validation Rules of Object System // Object Systems – 2013: Proceedings of the Seventh International Theoretical and Practical Conference (Rostov-on-Don, 10-12 May, 2013) / Edited by Pavel P. Oleynik. - Russia, Rostov-on-Don: SI (b) SRSTU (NPI), 2013. 14-17pp. (In Russian), http://objectsystems.ru/files/2013/Object_Systems_2013_Proceedings.pdf

[10] Oleynik PP. Domain-driven design the database structure in terms of metamodel of object system // Proceedings of 11th IEEE East-West Design & Test Symposium (EWDTS'2013), Institute of Electrical and Electronics Engineers (IEEE), Rostov-on-Don, Russia, September 27 – 30, 2013, pp. 469-472.

[11] Oleynik PP. The Elements of Development Environment for Information Systems Based on Metamodel of Object System // Business Informatics. 2013. №4 (26). – pp. 69-76. (In Russian), http://bijournal.hse.ru/data/2014/01/16/1326593606/1BI%204(26)%202013.pdf

[12] Oleynik PP. Domain-driven design of the database structure in terms of object system metamodel // Object Systems – 2014: Proceedings of the Eighth International Theoretical and Practical Conference (Rostov-on-Don, 10-12 May, 2014) / Edited by Pavel P. Oleynik. – Russia, Rostov-onDon: SI (b) SRSPU (NPI), 2014. - pp. 41-46. (In Russian), http://objectsystems.ru/files/2014/Object_Systems_2014_Proceedings.pdf

[13] Oleynik PP. Using metamodel of object system for domain-driven design the database structure // Proceedings of 12th IEEE East-West Design & Test Symposium (EWDTS'2014), Kiev, Ukraine, September 26 – 29, 2014, DOI: 10.1109/EWDTS.2014.7027052 http://dx.doi.org/10.1109/EWDTS.2014.7027052

[14] Oleynik PP, Kurakov Yu I. The Concept Creation Service Corporate Information Systems of Economic Industrial Energy Cluster // Applied Informatics. 2014. №6. 5-23 pp. (In Russian).

[15] Oleynik PP. To a question of need of design of hierarchy of atomic literal types for the object system organized in RSUBD//Information technologies and their appendices. IX International scientific and technical conference: collection of articles. – Penza: RIO PGSH, 2008. – pp. 201-205.

[16] Oleynik PP. The organization of hierarchy of atomic literal types in the object system constructed on the basis of RSUBD//Programming, 2009, № 4. – pp. 73-80.

[17] Oleynik P.P. Implementation of the Hierarchy of Atomic Literal Types in an Object System Based of RDBMS // Programming and Computer Software, 2009, Vol. 35, No.4, pp. 235-240. http://dx.doi.org/10.1134/S0361768809040070

[18] Oleynik PP. Unified Model for Testing of Tools for Object-Oriented Application Development // Object Systems – 2014 (Winter session): Proceedings of IX International Theoretical and Practical Conference (Rostov-on-Don, 10-12 December, 2014) / Edited by Pavel P. Oleynik. – Russia, Rostov-on-Don: SI (b) SRSPU (NPI), 2014, 25-35 pp. (In Russian), http://objectsystems.ru/files/2014ws/Object_Systems_2014_Winter_session_Proceedings.pdf

Received on 22-11-2015 Accepted on 08-12-2015 Published on 30-12-2015

http://dx.doi.org/10.15379/2410-2938.2015.02.02.02