International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 15 (2015) pp 35229-35238

Domain-Driven Design of Information System for Queuing System in Terms of Unified Metamodel of Object System

Pavel P. Oleynik, PhD

System Architect Software, Aston JSC
Shakhty Institute (branch) of Platov South Russian State Polytechnic University (NPI) Rostov-on-Don, Russia
xsl@list.ru

Nikolay V. Kuznetsov, PhD

Advisor of CEO, Kaskad JSC, Director of the Center for Institutions of Innovation Economy Development, Financial University under the Government of the Russian Federation, Moscow, Russia,
nkuznetsov@outlook.com

Edward G. Galiaskarov and Ksenia O. Kozlova

Ivanovo State University of Chemistry and Technology Ivanovo, Russia
portugaled@yandex.ru *and* ksu932011@mail.ru

Abstract
The paper presents an example of the implementation of an information system for queuing system that is designed on the basis of the metamodel of object system developed by the authors. The metamodel lies at the heart of construction of SharpArchitect RAD Studio. Using the metamodel is universally and significantly simplifies the process of developing information systems at different levels of complexity.

Keywords—DDD; Object System Metamodel; Object-Oriented System; Information System; Database

I. INTRODUCTION

Domain-driven design (DDD) is an approach to software development for complex needs by connecting the implementation to an evolving model. This article presents an example of the implementation of an information system that is designed on the basis of its own object system metamodel.
The structure of this article is as follows. Section 1 provides an overview of existing work on the development of metamodels of object systems. Section 2 presents the key requirements (optimality criteria) for the developed information systems to which it must meet. Section 3 presents the metamodel used in this work and describes key metaclasses and relationships between them. Section 4 contains the domain model of information system of a beauty salon, which meets the specified requirements and described the relationships between the classes. In conclusion, there are the findings and directions for further development of work.

II. REVIEW OF EXISTING PUBLICATIONS

This article describes the domain-driven design (DDD) of information system of a beauty salon in terms of unified metamodel of object system. The domain of a beauty salon was chosen precisely because this topic is closest to one of the authors and she is an expert in the area. In general the domain does not have a key value as metamodel is unified and can be used in any application databases.
The idea of the design based on the metamodel is not new since metamodels are used everywhere. Works [1-3] represent the metamodel of object database compliant with the ODMG standard. The SQL:2003 standard includes metamodel that describes the object extensions of SQL [1, 3-4]. The graphical modeling language UML which is often used in the design of modern applications has the standard which governs its own metamodel described in [3, 5].
Practically all the metamodels have some disadvantages. Each author was trying to solve the problem own forces. Thus, in [3], the author has developed its own metamodel for implementing object database.
Metamodel does not exist independently of the rest of the application, and serves a certain purpose. So in [6-7] metamodel is used to facilitate the process of domain-driven design.
Metamodel is also used in the model transformations. So in [8-9] the principles of model transformations from the UML metamodel in the models of languages developed by the authors are described.
Mechanisms for evaluating the software quality based on metamodel with the introduction of various metrics are considered in [10-11]. Publications [12-13] are devoted to questions of the expansion of existing metamodels by adding new elements. In [1], the author offers his own hierarchy of atomic literal types, which can be used in any object system and built thanks to the author's experience.

III. THE OPTIMALITY CRITERIA FOR INFORMATION SYSTEM QUEUING SYSTEM

The development of any software begins with the definition of optimality criteria (CO), which are the requirements for implementing the system. For the described application the optimality criteria (CO) are:

1. The information system should be managed as a separate beauty shop, and a network of salons. This will simplify the process of personal training and deployment of a software product.
2. It is necessary to implement a mechanism to preserve the history of changes in prices of goods and services. Many existing information systems that automate the activities of beauty salons, keep only the last actual price, on the basis of which is calculated the cost of a visit to the master. This approach leads to a number of problems when it is necessary to analyze the performance of the organization and the formation of aggregates reports.
3. It should be possible quantitative account of the goods in the context of warehouses and salons. Many systems do not offer this, as a result the financial statements are distorted and profit increases unnecessarily. In our system, it is necessary to make a number of documents reflecting the process of receipt of goods to warehouses, inventory results and the actual use of the goods. This will automatically build a trial balance for any day.
4. Implement leave records, absenteeism records, compensatory holiday records. This will allow to calculate the salary for each employee for any period of time, with a detailed report.
5. Implement different mechanisms to charges for services rendered. It is necessary for the calculation of the employee's salary and gives a complete picture of the costs of a beauty salon.
6. Provide registration certificates in the context of salons and clients. It will assess the projected profit in the future and will allow the owner of the salon strategically to plan its development.
7. Implement a mechanism of formation of working hours and tracking the compensatory holidays / absenteeism / sick leaves.

Let us proceed to consider the implementation of the described system. But first let us consider briefly a unified metamodel of object system.

IV. UNIFIED METAMODEL OF OBJECT SYSTEM

In this section, we briefly review the metamodel used in a unified development environment for the rapid development of enterprise information systems, SharpArchitect RAD Studio [14]. In [6-7, 15-18] the full class diagram of metamodel was presented and detailed assignment of classes were described. Here we consider only the relevant parts for this article. Fig. 1 shows a fragment of a unified metamodel of object system with display of the key associations that are important for further discussion.

Consider some of the key hierarchies of metaclasses. Fig. 2 is a diagram of the basic metaclasses used to represent different kinds of classes applied to describe the entity classes which are presented in the domain model.

An abstract metaclass Class is the root of the hierarchy. It has two inherited classes: 1) InheritableClass is used to represent metaclasses that can be inherited, i.e. support inheritance; 2) NotInheritableClass is used to represent metaclasses that can not be inherited. Metaclass Enum allows submit an enumeration or a set of values of a simple type.

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

Abstract metaclass ComputationalClass<TBaseClass> is the base for all calculated metaclasses ie those classes instances of which are not stored in the database and are computed 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 classes and represent the instances of the metaclass DomainClass).

Metaclass MethodParameterClass is used to represent a Parameter Class of methods. A design pattern called Parameter object, the essence of which is the transfer of a set of parameters in the method as a single object (an instance of the metaclass) is implemented in SharpArchitect RAD Studio.

Abstract metaclass CodeComputationalClass<TBaseClass> is the base for calculated metaclasses implemented using code in the language of C#. QueryClass is metaclass of query allowing to form the result based on queries to the database (usually based on Linq-object request but SQL-lines are also possible). HelperClass is used to represent the auxiliary metaclasses that can be displayed in the user interface and used for internal purposes in the implementation of business logic.

Now consider the metaclasses used to describe the class attributes shown in Fig. 3.

Root abstract metaclass representing an attribute is AbstractAttribute. Classes inherited from VirtualAttribute are used to represent attributes that were not created by the developer of 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 to describe the attributes that are system and are presented at the language of C#. Metaclass GeneratedAttribute is used to represent attributes automatically generated by the system. For example, if inheriting from the base tree class an attribute Node, which allows to get the child nodes and thus to form a hierarchical structure, are automatically added.

An abstract base metaclass ConcreteAttribute is used for presentation of attributes whose values can be defined by the user. Since the system is implemented in the language of C #, when saving values in the database the data types of this language are used. To describe this moment parameterized

metaclass TypedAttribute<TDefaultValue> was added. TypeAttribute is used to represent the properties whose values can store a reference to the data type of the C# language. Metaclass ClassedValueAttribute<TValueClass, TDefaultValue> is used to represent attributes whose values are the instances of the different instances of entity classes present in the domain. Metaclass NotInheritableClassedValueAttribute<TValueClass, TDefaultValue> retains instances of non-inherited classes. For example, EnumAttribute inherited from the one described is used to store values of enumerations / sets.

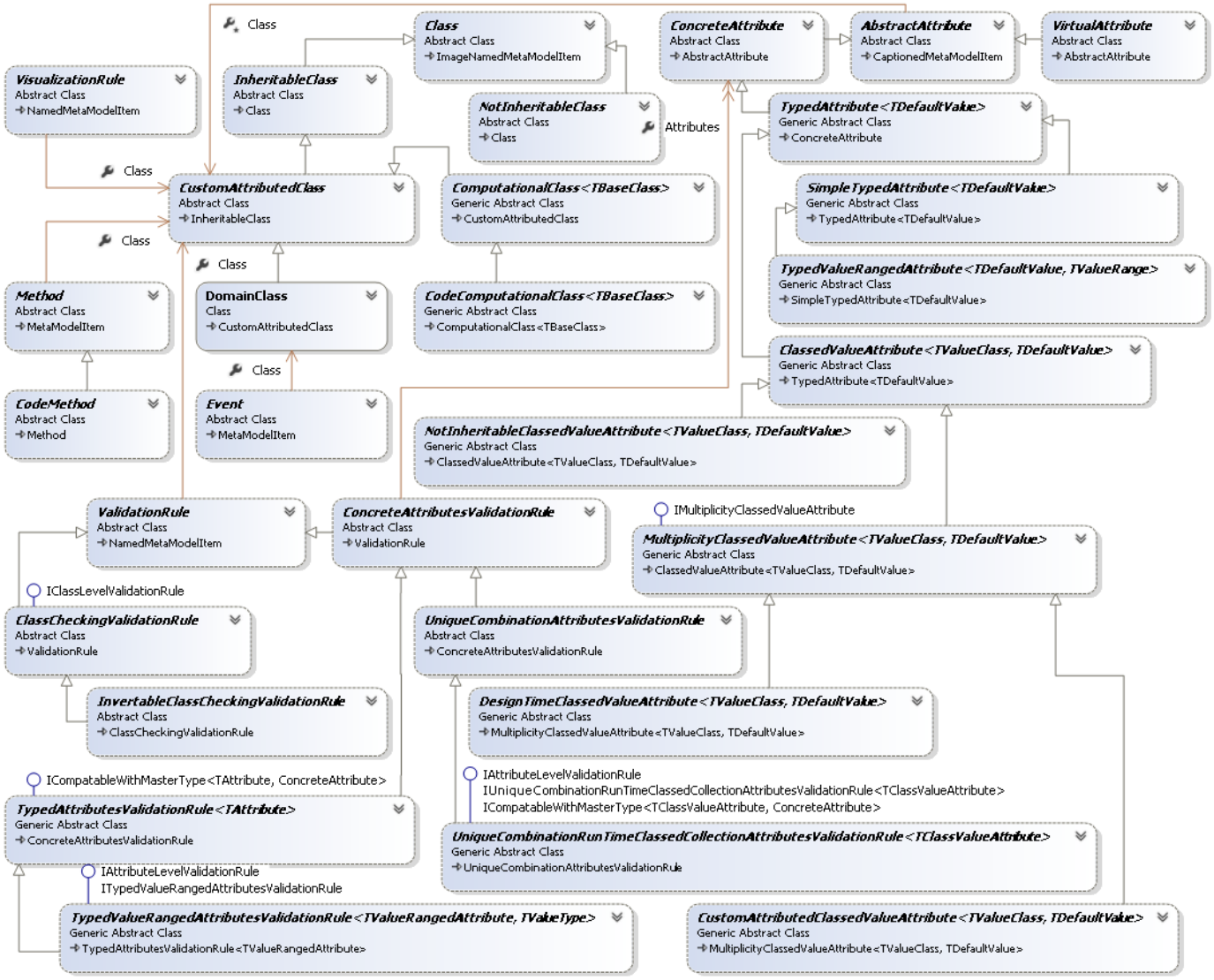

Fig. 1. **Fragment of a unified metamodel of object system**

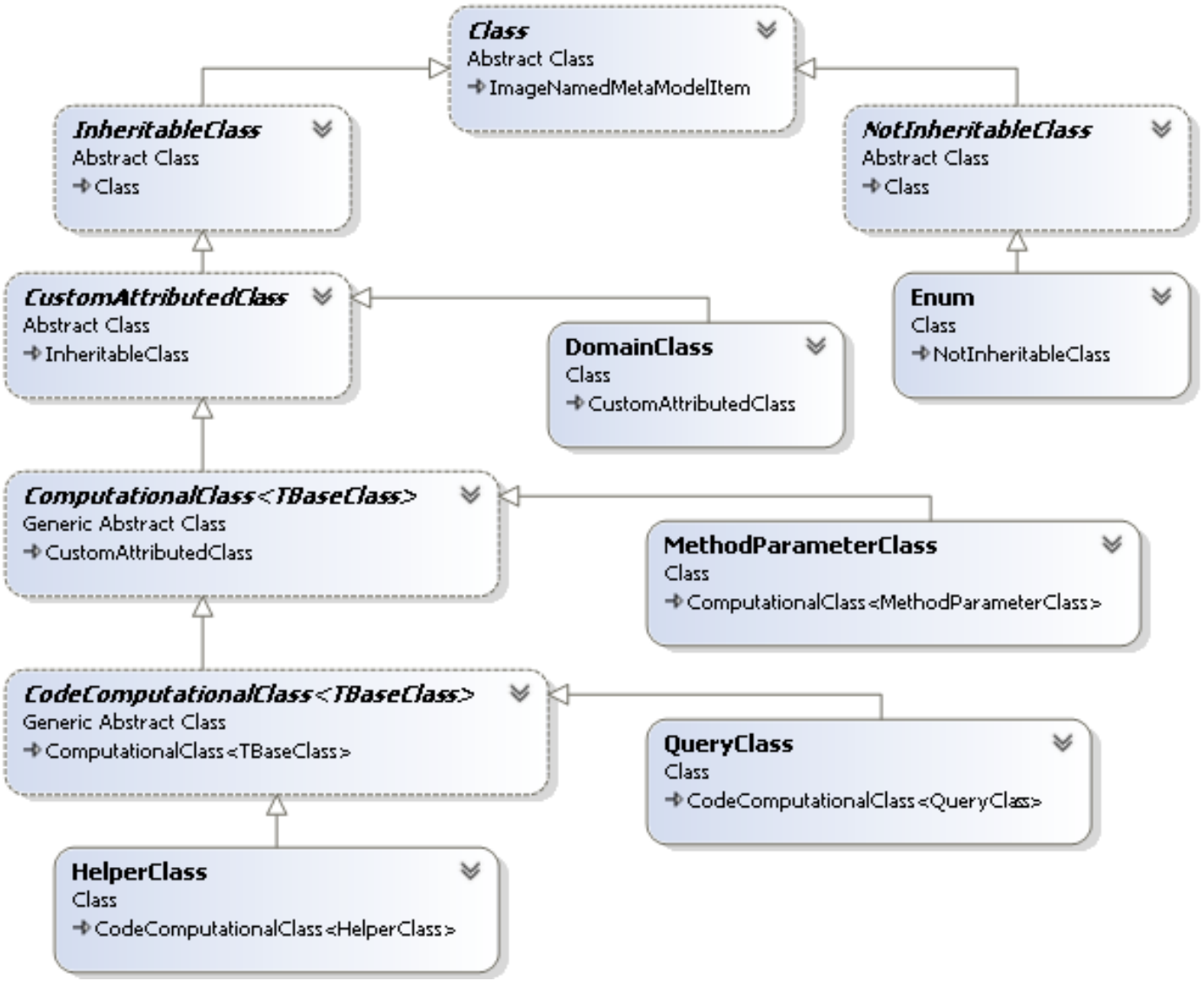

Fig. 2. **Basic metaclasses used to represent the entity classes of the domain model**

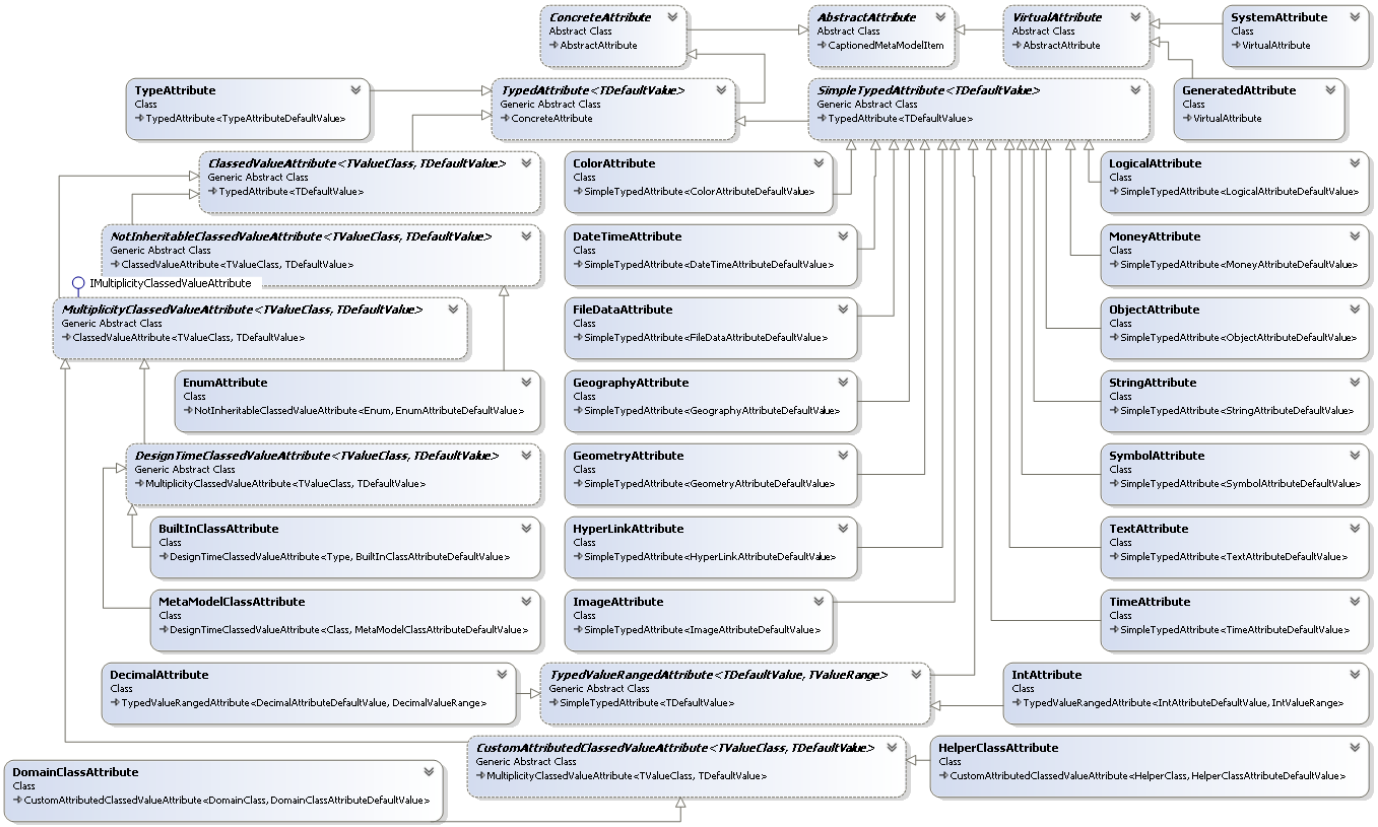

Fig. 3. **Basic metaclasses used to represent the attributes of classes**

Abstract metaclass MultiplicityClassedValueAttribute<TValueClass, TDefaultValue> is used to represent the values of the attributes that can store not only atomic values but also a collection of values. Metaclass DesignTimeClassedValueAttribute <TValueClass, TDefaultValue> allows saving a reference to instances of design time. So BuiltInClassAttribute is used to store objects of classes in the metamodel implementation in SharpArchitect RAD Studio. In its turn MetaModelClassAttribute saves information about the class metamodel of the application domain. Both described metaclasses allow manipulate metamodel at the moment of runtime. A similar approach is used in many modern programming languages supporting an extensive meta-information. So in C # there is a technology of reflection, which allows realizing such things.

Metaclass CustomAttributedClassedValueAttribute <TValueClass, TDefaultValue> is used to store instances of classes with attributes. The system has two child classes: 1) DomainClassAttribute saves a reference to the instance of a domain entity described in the metamodel using an instance of domain metaclass. Attribute of this type is used for the organization of the association relationships and serves to represent the relations with the object design of the domain. Metaclass HelperClassAttribute allows saving references to instances of helper classes.

Metaclass SimpleTypedAttribute <TDefaultValue> is abstract and serves the root of all the attributes to save the atomic literal value. All of this hierarchy is the result of many years of work, the premise of which and the intermediate solutions have been described in [1]. Metaclass ColorAttribute is used to store the color in the format of RGB. LogicalAttribute is used for storing Boolean values (true and false). Metaclass DateTimeAttribute is used to save the date-time values. If you want to present time only you should use TimeAttribute. There is metaclass MoneyAttribute for submission to the money attribute in the hierarchy. FileDataAttribute is used to save files of various formats. Attribute of type ObjectAttribute should be used to store any type of object. This approach is similar to the use of type object in C#. Attributes of types GeographyAttribute and GeometryAttribute are used to save the geographical coordinates and geometric objects, respectively. Metaclasses StringAttribute and SymbolAttribute are used to represent character strings and individual characters respectively. If you want to save the text of unlimited length and with formatting you should use TextAttribute. Metaclass HyperLinkAttribute is used to represent hyperlinks to various resources. ImageAttribute is used for storing graphics (pictures, photographs and the like). Parameterized abstract metaclass TypedValueRangedAttribute <TDefaultValue, TValueRange> is used to represent atomic values, which may be within a certain range of values specified by the relevant enumeration (parameter TValueRange). Inherited metaclass IntAttribute can be used to store integer values and DecimalAttribute can be used to represent fractional values.

To implement the behavior in SharpArchitect RAD Studio different syntactic constructions and metaclasses are used.

Class methods, metaclasses of which are shown in Fig. 4 are the most commonly used.

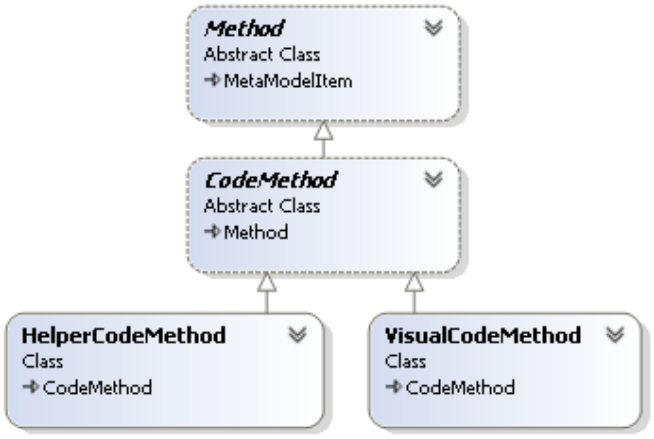

Fig. 4. **Basic metaclasses used to represent the class methods**

Method is the root abstract metaclass of method. Currently, the system supports only the methods implemented in the form of program code and submitted as instances of metaclasses inherited from CodeMethod. Metaclass VisualCodeMethod will create a visual method, which is displayed to the user in the form of a graphic element in the interface. HelperCodeMethod is a helper method that is used to call other methods and properties and is not involved in forming the interface.

Events are an integral part of behavior used in the development of object-oriented applications. Fig. 5 shows metaclasses allowing to describe the various events of objects.

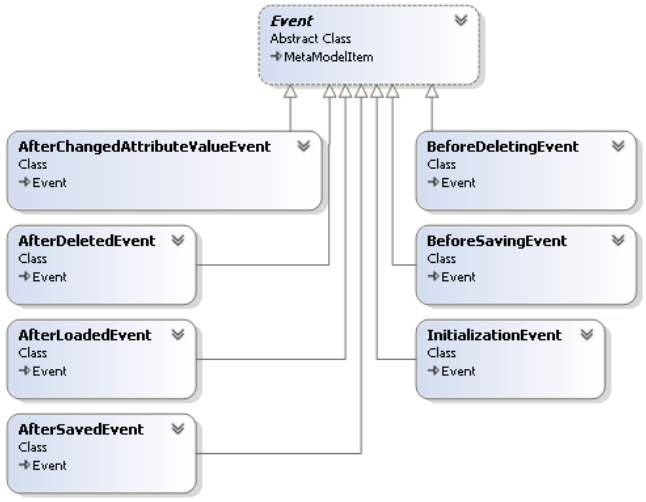

Fig. 5. **Basic metaclasses used to represent events**

The hierarchy is very simple. Event is the root abstract metaclass represented an event. AfterChangedAttributeValueEvent describes events of changing the attribute values and has in its composition, not only the name of the changed attribute, but also the possibility of obtaining the values before and after the change. Event AfterDeletedEvent is called after the removal of the object. Note that the object is not physically removed from the database, but only marked as deleted. This is due to the possible presence of links to other objects and the need to provide an opportunity to recover accidentally deleted object. Exactly after the installation of this label the event is called. Metaclass AfterLoadedEvent allows to create an event handler that occurs after loading the object from the database. Metaclass AfterSavedEvent describes an event that occurs after saving the object in the database. Metaclass BeforeDeletingEvent is used to represent an event that occurs before the removal of the object. If you want to perform certain actions before saving you must create an instance of the metaclass BeforeSavingEvent. The code of the new object initialization is performed in the object of type InitializationEvent.

At present an essential step in the development of any large system is to write validation rules that are checked at a certain time (usually while maintaining the object) and confirm the integrity and consistency of data. Hierarchy of validation rules is shown in Fig. 6. ValidationRule is the root abstract metaclass. ClassCheckingValidationRule is directly inherited from it and allows to describe the rules for checking the conditions at the class level (because it implements the interface IClassLevelValidationRule). Metaclass InvertableClassCheckingValidationRule enables to describe invertible conditions. Implemented metaclass CriteriaCheckingValidationRule allows to set the logical test condition as a string and specify a set of attributes that violate this condition. ConcreteAttributesValidationRule describes the validation rules that involve the certain class attributes inherited from ConcreteAttribute. Derived metaclass RequiredAttributesValidationRule enables to specify which class attributes are mandatory and can not contain null values. The derivatives from the parameterized abstract metaclass TypedAttributesValidationRule <TAttribute> allow to describe validation rules that apply only to the attributes of a certain type (of the type specified in TAttribute). Metaclass RegularExpressionAttributesValidationRule is used to specify a regular expression that values of the string attributes (StringAttribute) must match.

Parameterized abstract metaclass TypedValueRangedAttributesValidationRule <TValueRanged Attribute, TValueType> is used to create validation rules of entering values in a certain range. Its children RangeDateTimeAttributesValidationRule, RangeDecimal AttributesValidationRule, RangeTimeAttributesValidation Rule, RangeIntAttributesValidationRule are used to specify a range of datetime, decimal, time and integer attributes, respectively.

UniqueCombinationAttributesValidationRule is the base abstract metaclass specifying a unique combination of attribute values. Its child UniqueCombinationSimpleAttributes ValidationRule is used when defining a unique combination of simple atomic values. Another child UniqueCombinationRunTimeClassedCollectionAttributesVali dationRule <TClassValueAttribute> is used for attributes that

represent a collection of values and allows to determine the unique attributes for each element in the collection. At the moment, there is only one inherited implemented metaclass UniqueCombinationDomainClassCollectionAttributesValidationRule, which allows setting a unique combination of values for attributes of the domain entities.

In big applications, there is the problem of highlighting of separate fields, columns or rows depending on their values. In addition, it is often necessary to hide or to deactivate some input fields or hide them from the form.

Hierarchy of metaclasses of visualization rules, shown in Fig. 7, solves this problem. VisualizationRule is the root base metaclass. ActionsVisualizationRule allows to describe the visualization rules for the various actions, such as the Save button, the Create New button, the Edit button, etc. Metaclass AttributesVisualizationRule describes the visualization rules for class attributes. Instances of MethodsVisualizationRule permit to define rules for visualization methods (Instances of VisualCodeMethod).

As can be 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. The IDE has been tested on different projects, described in [18, 19].

V. THE DOMAIN MODEL OF INFORMATION SYSTEM OF QUEUING SYSTEM

Fig. 8 shows the domain model of beauty salons built on the basis of metamodel discussed above and taking into account the selected criteria of optimality. Notice that all obtained classes have been described in the metamodel in the form of domain classes (metaclass DomainClass, Fig. 2). At the same time for the implementation of the domain entities the programming language of C# and syntactic construction of interface are used. Generation of interfaces is performed on the basis of metaclasses instances stored in the database. Consider in detail the composition and structure of the classes presented on the diagram.

Nomenclature of goods and services is provided with the class hierarchy of Commodity (is the hierarchy root, which is involved in various associations), of Service (for description of service), of Product (for definition of goods sold). Please note that classes of Service and Product are inherited from both class of Commodity and class of IBaseRunTimeTreeNodeDomainClass, which is the system domain class that is used to describe a hierarchical (tree) structure.

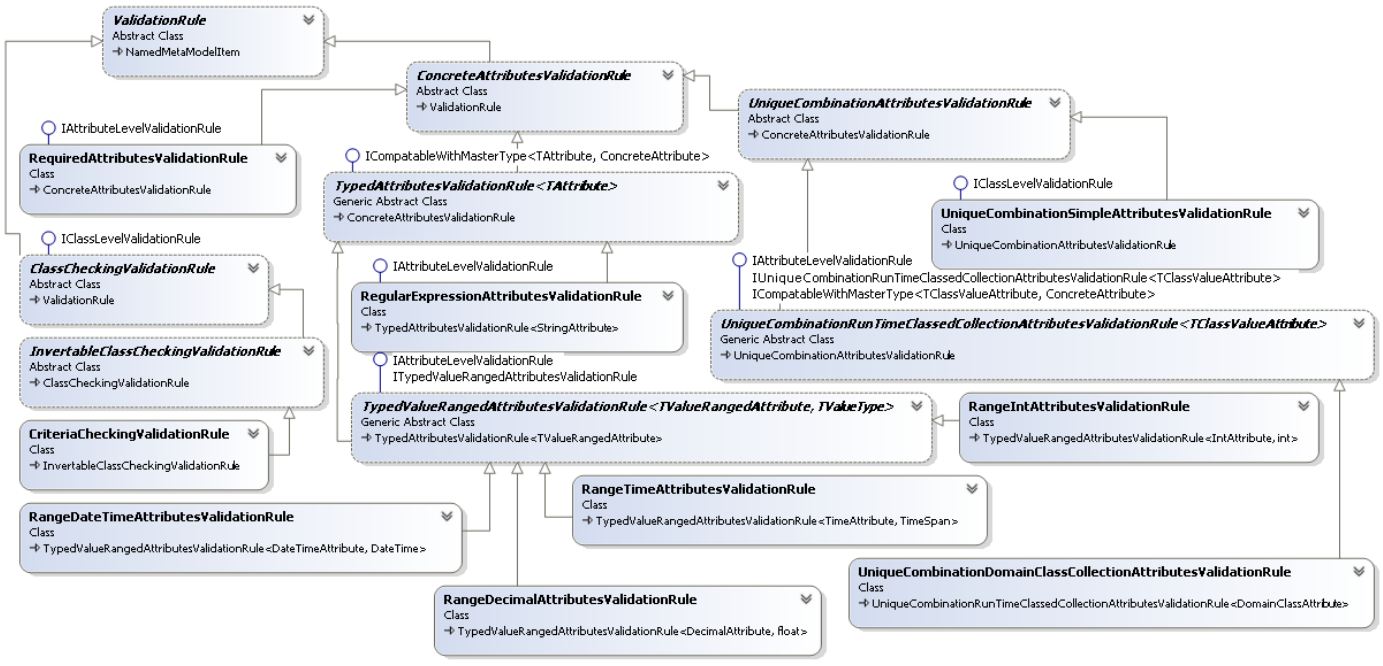

Fig. 6. **Basic metaclasses used to represent the validation rules**

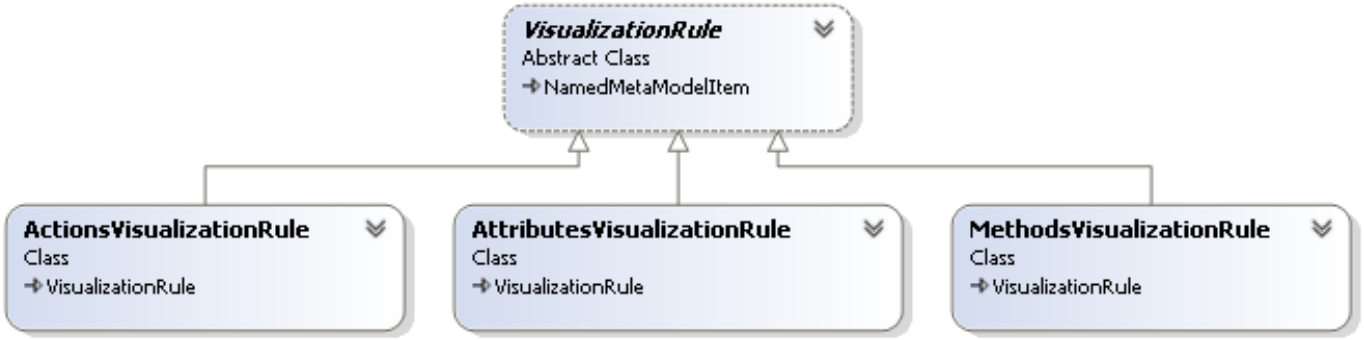

Fig. 7. **Basic metaclasses used to represent the visualization rules**

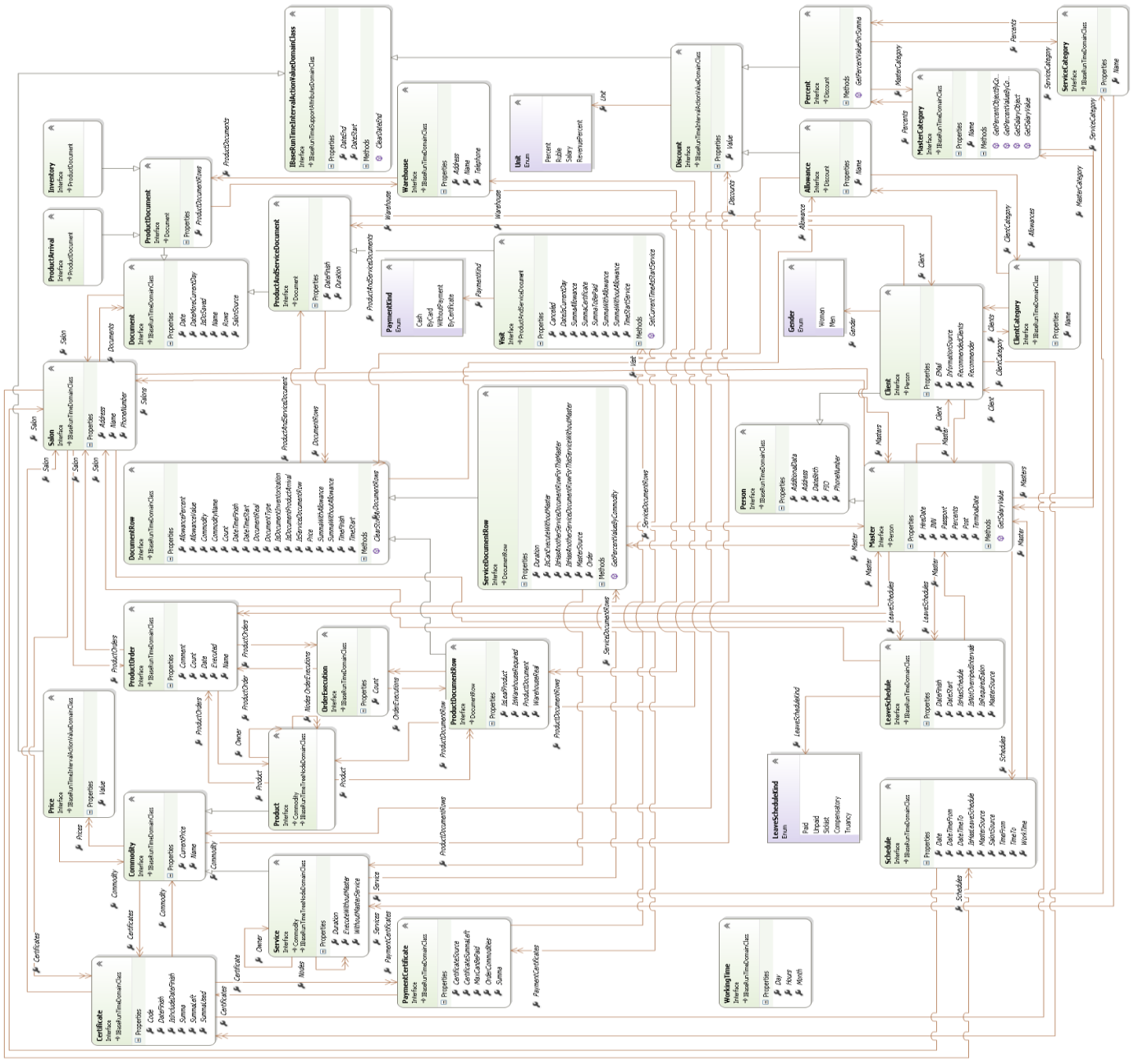

Fig. 8. **Domain model of information system of a queuing system, built in terms of the metamodel of object system**

Thus, the system supports multiple inheritance, which is realized using the interface of C# language, the concept is discussed in detail in [20-23].

Class Price is used to save the history of changes in prices of goods and services. It is inherited from IBaseRunTimeIntervalActionValueDomainClass containing the attributes DateStart and DateFinish, which are used to store the operating range of prices.

Information about salons is stored as instances of class Salon. Class hierarchy, the root of which is class Document is used for organization of document interchange. From a logical perspective, all documents can be divided into two groups. The first group includes the documents, which may contain strings both goods and services. Class ProductAndServiceDocument is the root of the hierarchy of such documents. Class Visit describing the record and visit of the client is derived from this class. The idea is when the client is recorded on getting a service, the document Visit is formed, and the planned date and time of the visit are indicated in it. Upon the occurrence of this time, it is considered that the client either came or was late that indicate the appropriate options.

The second group includes documents, which contain only rows of products available in the salon. Class ProductDocument is the root of the hierarchy and has two subclasses: 1) class ProductArrival describes the arrival of goods at the warehouse; 2) class Inventory is an inventory of existing stocks in warehouses. Class Warehouse is used to represent the existing warehouses.

Class Master is used to represent information about the masters of salons. Class Client is a representation of the clients. Both classes are inherited from the abstract class Person, which comprises common properties.

To describe the discounts given to customers Allowance class is used. For representation of interest earned by masters for services rendered, class Percent is applied. Both of these classes are inherited from the root class Discount, which in turn is inherited from IBaseRunTimeIntervalAction ValueDomainClass, described earlier. In this case, the discount is assigned not to a specific master or client, and the master category or the client category, respectively. This approach reduces the time to configure the system and is implemented by classes MasterCategory and ClientCategory, respectively. Also, services can be grouped into categories using ServiceCategory.

Each master can leave an application for goods needed by instances of the class ProductOrder. Execution of such applications is analyzed using the class OrderExecution.

The salon can issue certificates for various services by creating instances of the class Certificate. The client uses them to pay for various services through the creation of instances of the class PaymentCertificate.

Work schedule is one of the key concepts on which depends the coordinated work of all beauty salons. Class Schedule is used to represent the working time of masters. Sick leave, compensatory leave and absenteeism of employees are represented by class LeaveSchedule. This information is used for automatic payroll preparation. At this point the instances of the class WorkingTime are also used. They provide information on working hours in each month. Thus it is seen that the domain model described above meets to the criteria of optimality specified previously.

CONCLUSIONS AND FURTHER RESEARCH

This article describes an example of domain-driven design for information system that automates the activities of beauty salons. At the moment, system is under a test operation in one of the salons of the Russian Federation. The system is developed in terms of a unified metamodel of object system described in detail in the relevant section of this article. As further development of the work the authors suggest the development of a formal mathematical apparatus describing applied domains and the development of UML-profile that facilitates the process of logical design of information systems in the framework of the proposed approach.

REFERENCES

[1] 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.

[2] Cattell R.G., Barry D.K. The Object Data Standard:ODMG 3.0, Morgan Kaufmann Publishers, 2000, 288p.

[3] Habela P. Metamodel for Object-Oriented Database Management Systems. Ph.D. Thesis // Submitted to the Scientific Council of the Institute of Computer Science, Polish Academy of Sciences, 2002, 142p.

[4] Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation), http://wiscorp.com/sql200n.zip.

[5] Iyengar S., Brodsky S. Metadata Integration using UML, MOF and XMI, November 2000, 88p.

[6] Oleynik P.P. 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.

[7] Oleynik P.P. 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

[8] Soon-Kyeong K., Carrington D., Duke R. A metamodel-based transformation between UML and Object-Z // Human-Centric Computing Languages and Environments. Proceedings IEEE Symposia, 2001, 112-119 pp.

[9] Rahim L.A. Mapping from OCL/UML metamodel to PVS metamodel //Information Technology, ITSim 2008. International Symposium, 2008, 1 – 8 pp.

[10] McQuillan J.A., Power J.F. A Metamodel for the Measurement of Object-Oriented Systems: An Analysis using Alloy // Software Testing,

Verification, and Validation, 2008 1st International Conference, 2008, 288 – 297 pp.

[11] Debnath N., Riesco D., Montejano G., Uzal R., Baigorria L., Dasso A., Funes A. A technique based on the OMG metamodel and OCL for the definition of object-oriented metrics applied to UML models // Computer Systems and Applications, 2005. The 3rd ACS/IEEE International Conference, 2005.

[12] Debnath N., Riesco D., Montejano G., Grumelli A., Maccio A., Martellotto P. Definition of a new kind of UML stereotype based on OMG metamodel // Computer Systems and Applications, 2003. Book of Abstracts. ACS/IEEE International Conference, 14-18 July 2003.

[13] Misbhauddin M. , Alshayeb M. Extending the UML Metamodel for Sequence Diagram to Enhance Model Traceability // Software Engineering Advances (ICSEA), 2010 Fifth International Conference, 22-27 Aug. 2010, 129 – 134pp.

[14] Oleynik P.P., 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).

[15] Oleynik P.P. 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

[16] Oleynik P.P. 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

[17] Oleynik P.P. 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

[18] Oleynik P.P. 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

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

[20] K.O. Kozlova, P.P. Oleynik. Experience of Using Multiple Inheritance in Modern Programming Languages // 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. – pp. 10-12. (In Russian), http://objectsystems.ru/files/2014WS/Object_Systems_2014_Winter_session_Proceedings.pdf

[21] Pavel P. Oleynik, Olga I. Nikolenko, Svetlana Yu. Yuzefova. Information System for Fast Food Restaurants. Engineering and Technology. Vol. 2, No. 4, 2015, pp. 186-191., http://article.aascit.org/file/pdf/9020895.pdf

[22] Pavel P. Oleynik. Using Multiple Inheritance in Modern Frameworks. Engineering and Technology. Vol. 2, No. 4, 2015, pp. 202-206., http://article.aascit.org/file/pdf/8960766.pdf

[23] Pavel P. Oleynik. Metamodel-Driven Design of Database Applications. Journal of Computer Science Technology Updates, 2015, Vol.2, No. 1, pp. 15-24., dx.doi.org/10.15379/2410-2938.2015.02.01.03, http://www.cosmosscholars.com/images/JCSTU-v1n1/JCSTU-V2-N1/JCSTU-V2N1A3-Oleynik.pdf