2. Елисеев Д.В. Балдин А.В., Тоноян С.А. Язык запросов к миварному представлению реляционных баз данных, содержащих архив информации из предыдущих кадровых систем // Инженерный журнал: наука и инновации. – 2013. – № 11 (23). С. 20.
3. Developing Time-Oriented Database Applications in SQL, Richard T. Snodgrass, Morgan Kaufmann Publishers, Inc., San Francisco, July, 1999, 504 pages.
4. История и актуальные проблемы темпоральных баз данных [Электронный ресурс] / Костенко Б.Б., Кузнецов С.Д., 2007 – Режим доступа: http://citforum.ru/database/articles/temporal/4.shtml, (Дата обращения: 18.03.2015)
5. Andreas Steiner. A Generalisation Approach to Temporal Data Models and their Implementations [Электронный ресурс], 1998 – Режим доступа: http://www.timeconsult.com/Publications/diss.pdf, (Дата обращения: 18.03.2015)
6. Елисеев Д.В., Балдин А.В. Обзор способов построения темпоральных систем на основе реляционной базы данных // Инженерный журнал: наука и инновации. – 2012. – №3. (3). С. 5 – 12
7. Дейт К. Введение в системы баз данных. – 8-е изд. – М.: Вильямс, 2006. – 1328 с.
8. Елисеев Д.В., Балдин А.В., Тоноян С.А. Анализ избыточности хранения темпоральных данных средствами реляционной СУБД // Инженерный журнал: наука и инновации. – 2014. – № 4 (28). С. 1.
9. Варламов О.О. Эволюционные базы данных и знаний для адаптивного синтеза интеллектуальных систем. Миварное информационное пространство. – М: Радио и связь, 2002.-286 с.
10. Варламов О.О. Основы многомерного информационного развивающегося (миварного) пространства представления данных и правил // Информационные технологии. – 2003. – № 5. – С. 42-47.
11. Адаптируемая модель данных на основе многомерного пространства [Электронный ресурс] / Елисеев Д.В., Балдин А.В.- Электрон. журн. – М.: «Наука и образование: электронное научно-техническое издание», 2010 – Режим доступа: http://technomag.edu.ru/doc/161410.html, свободный, (Дата обращения: 18.03.2015)
12. Елисеев Д.В. Методика обработки темпоральной реляционной базы данных в миварном пространстве: 05.13.17: автореф. … дис. ктн; МГТУ им. Н.Э. Баумана. – М., 2011. – 16 с.
13. Алгебра многомерных матриц для обработки адаптируемой модели данных [Электронный ресурс] / Елисеев Д.В., Балдин А.В.- Электрон. журн. – М.: «Наука и образование: электронное научно-техническое издание», 2011 – Режим доступа: http://technomag.edu.ru/doc/199561.html, свободный, (Дата обращения: 18.03.2015)

UDK 004.04

OBJECT MODELING OF SUBJECT DOMAINS

Ekaterina M. Lavrischeva, Doctor of Physical and Mathematical Sciences, Professor MIPT, Chief Researcher of ISP RAN, Moscow, Lavrischeva@gmail.com

Introduction

On verge of 1980s, object-oriented programming approach was introduced by G. Booch, changing the operating programming process. At the time, software industry encountered the *complication crisis*. Not only Brooch's theory was necessary to end the crisis, but *technology* as well. Object-oriented programming (OOP) is targeted not only towards improvement of the traditional approach that induced the complication crisis of systems developed with the help of PL; instead, it created a new style for programming systems by modeling subject domains (SD) with objects and their interfaces. OOP languages have surfaced for various object types, routines and data types These languages partially use the formal mechanisms of specifications programs (RAISE, RCL, VDM, Dijkstra, Hoare et al.) [1–5]. Derivatives of the elements of these languages became the general purpose resources for development of large-scale systems. In addition, it was

appeared systems offering (DCOM, CORBA, DCE RPC, etc.), which given the basis of formal determination of object model (OM) and their dispatch by the object request broker (ORB). New systems created with the help of real and library objects lower complexity of software programs (SP), replenish ready-made objects and reduce certain difficulties in case of changes data types and functional objects [6].

Now the modeling of domain by the formal specification, ontological tools OWL, ODM and models development (MDD, MDA, GDM and so on) were formed. A formal vehicle is oriented to the presentation of domain as a system of objects, signature of operations set (union , intersection ∩, adding ⊕, etc.) and predicates theory from axioms, theorems and logical assertions. The paradigm modelling SP from objects is created from the formal specifications in OKM method [7, 8]. In this method design begins from decomposition of domains by objects, determinations of their notation by denotes and concepts (theory Frege), and also presentations the OM as a graph. Each object will have behaviors which uniquely belong to this object The theoretical and applied conceptions of OKM consists of base notions and new positions of improved theory of design SD from objects of functional, interface data and isomorphism reflection of SD function of objects to the program components [9–11].

1. Mathematical Design of Objective Model

Object design theory is built with the use of base notions of formal specification, set theory and class theory of G. Booch, Frege triangle and CORBA object model, utilizing the following principles [1-3]:

- All essences of the SD are objects;
- Each object is a unique element;
- All objects are determined at a certain abstraction level and are ordered according to their relations;
- Object interoperability with help the of interfaces.

An object is singled out using object-oriented analysis, with mathematical terms for description and clarification of object methods in the OM being created.

According to Booch, «object-oriented approach = objects + inheritance, polymorphism, encapsulation»; OM also encompasses object classes and their relations (aggregation, associations, specializations, instantiation so on), as well as their behavior.

Object in SD is a named part of actual reality with a certain abstraction level; a notion structure according to Frege triangle (denotation, sign, and concept). Each object (O) belongs to the set of objects $O= (O_1, O_2. O_n)$, where $O_i = O_i (Na_i, Den_i, Con_i)$, Na_i is a sign, Den_i is a denotation, Con_i is an object concept, and $Con_i = (P_{i1}, P_{i2} ,..., P_{is})$ is determined upon a set of predicates P_i [8, 9].

Axiom 1. The subject domain designed with objects is an object itself.

Axiom 2. The subject domain being designed may be an object within another subject domain.

When designing the subject domain, each object gets at least one property or description, semantics allowing its unique authentication among the set of all objects and to the set of predicates of properties and relations between objects.

Object property is defined on the set of objects belonging to the SD with the unary predicate with return value depending on its external and internal properties. Description is an aggregate of properties (in form of predicates) subjected to the condition of acceptance of truth value by no more than one predicate from the collection of external and internal descriptions. **Relation** is a binary predicate that returns truth on each pair of objects in the set. The basic types of mutual relations are as follows:

1. Set – set;
2. Element of a set – element of a set;

3. Element of a set – set;
4. Set – element of a set.

These relation types correspond to operations: *generalization, specialization, aggregation, association,* classification *and instantiation.* Types 3), 4) are *subsumption* relation (*IS–A*) and part-whole relation (*PART–OF*), respectively.

OM Modeling Levels

SD model is designed on four levels:

- Generalizing for determining SD base notions without considering of their essence and properties;
- Structuring for ordering objects in the OM taking into account relationships between them;
- Characterization for forming concepts of objects on the base of them properties and descriptions;
- Behavioral level for descriptions of conduct depending on events (such as time).

In accordance with the **generalizing** level an object is considered a mathematical notion, as a class from the point of view of von Neumann–Bernays–Gödel set theory: $O= (O_0,, O_1, O_2. O_n)$, with O_0 being an object in the subject domain. A set of base functions is formed at this level, related to decomposition or composition changes to object denotations and concepts, performed by increasing or decreasing object quantity, as well as expansion or narrowing object concepts. These changes are subjected to the set rules and terms that ensure correctness of function implementation. For the set $O= (O_0, O_1, O_2. O_n)$, the object relations hold:

$$\forall i \ (i > 0) \Rightarrow (O_i \in O_0) . \quad (1)$$

The **structural** level defines such notions as class, class instance, abstract class, etc. The set of objects is ordered and each of objects can be presented as a set or an element of a certain set. That is, expression (1.1) is transformed into

$$\forall i > 0 \ \exists j \geq 0 \ (i \neq j) \wedge (O_i \in O_j) \quad (2)$$

It determines the "part–whole" relation, instantiation and aggregation.

In accordance with the **characteristic** level, for each of objects a corresponding concept is formed. If $O'= (O_1, O_2. O_n)$ is a set of objects SD, and $P'= (P_1, P_2, P_r)$ is a set of unary predicates related to properties of SD objects, concept of the object O_i is a set of assertions, built on the basis of predicates from P' that are true for the object. That is, the concept $Con_i = \{P_{ik}\}$, if a condition $P_k (O_i) = true$, where P_{ik} is the assertion for the object O_i according to the predicate P_k. Following these rules, the properties of objects are determined with the subsumption relation. Expression $A = (O', P')$ determines the algebra system of object concepts O' and predicates P' with operations:

- 0–ary operations that correspond to constants;
- Unary operations that correspond to the properties of objects;
- Binary operations that correspond to intercommunications between pairs of objects.

Predicates must meet specific conditions:

- Number of predicates suffices the conceptual design of the subject domain using its objects;
- Each predicate, its type and signature meets the essence of the corresponding object.

Axiom 1.Every object of the SD process has at least one feature or property, which equates to the set of objects.

In obedience to the behavioral level, a sequence of object states and processes is determined in order to reflect transitions between states. Intercommunications between objects are formed on the basis of binary predicates, which are related to the properties of SD objects, and are detailed to implement interoperability between states of objects.

According to the concept described above, *class* is an object that reflects a certain set; instance of the *class* is an object belonging to a certain set, which is a class; *joint class* is a set equal to the direct sum of several other sets; crossbred class is a common part of several other sets;

aggregated class is a subset of the cross product of several other sets. If an object is an element of another object, it is determined by the set. However, not every object is necessarily an element of another class. For example, an object corresponding to the entire subject domain in the OOM is not an element of any other object in the model. Definition of objects is formulated under the condition: each object is a set or an element of a certain set. Object ordering is performed taking into account affiliation by using sets of natural numbers.

Algebra for object-oriented analysis of the subject domain is

$$\Sigma = (O', I', A', P'), \tag{3}$$

where $O'=(O_1, O_2, .., O_n)$ is a set of objects; $I= (I_1, I_2 ., I_n)$ is a set of interfaces for O'; $A' =(A_1, A_2 .., A_n)$ is a set of operations on elements of a set O; $P = (P_1, P_2 ,.., P_r)$ is a set of predicates that determine properties of object concepts. Each of operations in A' possesses certain priority and arity, and also related to the corresponding acceptable descriptions of object concepts and operations from the set A' = {decds, decdn, comds, comdn, conexp, connar}. That is, decds, decdn are decomposition operations, comds, comdn are compositions, conexp, and Conner is narrowing [10].

Theorem 1. Set of operations *A'* for the algebra on the objects *O'* is a system of actions in relation to the functions of four-level object presentation of the object-oriented model. Operations of object analysis are:

- Specification of object, as a *class, class instance*, etc.;
- Operations above essences: *0–ary*, unary, *binary;*
- Interrelation of *generalization*, specialization*, aggregation, classification, instantiation;*
- Operations of object behavior together with communications between descriptions and the time of their existence in the OOM.

The SD model may be represented by an object graph $G =\{O, I, R\}$, defined on the set of objects *O*, interfaces *I* and relations between objects *R:*

- Set of vertices *O* replicates one-to-one relationships between objects in the subject domain;
- Each vertex corresponds to at least one interface $I_k \in I$ and relationships from the set *R* according to certain rules
- There exists at least one vertex with dual set –object status that reflects the entire domain.

The set of objects-functions *O* is related to implementation methods for objects in the subject domain, which communicate between themselves via interface objects from the set *I*. That is, vertices from G are objects of two types – functional objects *O,* and interface objects *I* (fig.1).

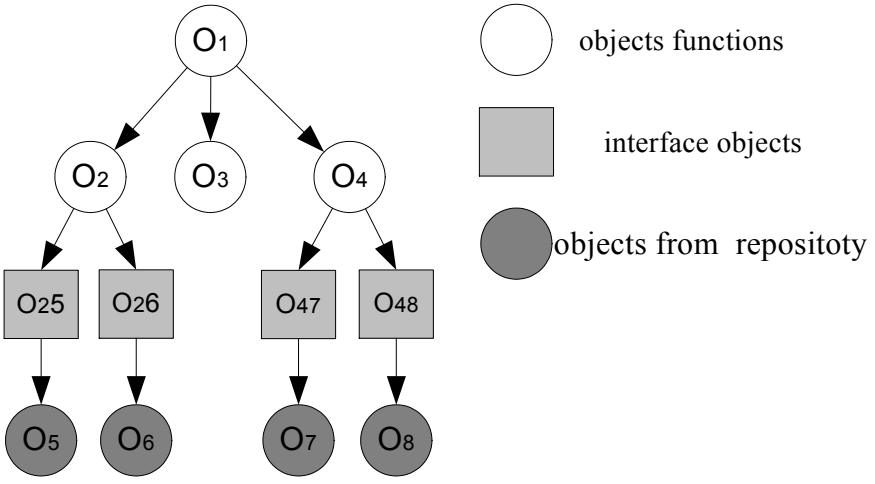

Fig. 1 – Object-interface graph for the OM

Interface objects contain data description that is passed by RPC, RMI, ORB requests with optional operations of data transformation to the proper format of the environment containing the

object implementation. The result of communication of a pair of objects in the graph (e.g., O_{25} and O_{47}) is a new interface object with the description of input (*in*) and output (*out*) parameters of a request or a communication protocol.

Axiom 2. Graph *G*, complemented with interface objects, is well-organized structurally (bottom-up) with regard to the control of fullness, surplus and removal of duplicate elements.

Objects may have several interfaces that can inherit interfaces of other objects, providing they provide services for the entire set of output interfaces.

The set of objects and interfaces of the graph is reflected by general or individual properties and descriptions of the object-oriented model. Verification of properties of objects is provided by the specific operations (classification, specialization, aggregation, etc.). Each operation is a pairwise comparison of the underlying object properties with their external characteristics. They are reliable in case the following condition holds: each underlying property is equivalent to the external property of object. If this condition does not hold, an element is removed from the set *O* and the graph.

2. Interfaces of objects

Object interfaces contain input (*In*) and output (*Out*) parameter specifications. The set of interfaces $I = \{In\ (O_k)\}$ is used by cooperating objects, whereas the set of output interfaces *Out* (O_k) reflects the obtained results. Specification of parameters In, Out for interface objects is performed with IDL language, which is a part of CORBA [12].

Objects in the OOM are defined by the following sets:

- $O_k \in O, In(O_k)$ – a set of input (*In*) interface objects;
- $O_j \in O, Out(O_j)$ – a set of output (*Out*) interface objects.

The result of co-operation between the two objects in the graph G is an intermediate object, with the set of input interfaces coinciding with the set of input interfaces of the accepting object, and the set of output interfaces corresponding to the set of output interfaces of the transmitter:

$$O_k = (Out(O_k), In(O_k)),\ O_l = (Out(O_l), In(O_l)),\ O_k \cdot O_l = (Out(O_k), In(O_l)) \quad (4).$$

Axiom 3. Object composition $O_k \cdot O_l$ is correct, if and only if the transmitter object provides a necessary service to the acceptor object: $\forall I_m \in In(O_k) \Rightarrow \exists I_n \in Out(O_l) \wedge I_m = I_n$.

Objects may have several interfaces that may inherit interfaces of other objects ($O_k \leftarrow O_l$); in this case the latter provides services for the set of output interfaces: $O_k \leftarrow O_l \Rightarrow Out(O_k) \subseteq Out(O_l)$

An inherited object delegates all its interfaces to the other object and possesses the following characteristics:

- Transitivity: $\forall O_{1,2,3} \in O : O_1 \leftarrow O_2, O_2 \leftarrow O_3 \Rightarrow O_1 \leftarrow O_3$;
- Reflectivity: $\forall O_k \in O \Rightarrow O_k \leftarrow O_k$.

Reflection of an object onto another object results in an interface consisting from the set of input interfaces $O_k[O_l] = O_k[In(O_l)]$ and output interfaces $O_k[O_l] = O_k[Out(O_l)]$.

An example of parameter specification for interfaces using the stub/skeleton-type broker is displayed. In interface type of data, that are passed through the call (RPC, RMI, ORB and others like that) parameters are described in IDL. These parameters answer data of objects-functions, which give by languages (C#, Vbasic, Pascal, etc.). Parameters of data of interface can be can be specificity as input (*In*), output (*Out*) and compatible (*Input*).parameter.

The example of parameter of interface mediator of type stub F_1 and skeleton F_2 specification in chart (Fig. 2) has a kind on Fig. 3 [7–10].

Despite input and output interfaces having different semantics, they share identical syntactic description. At the formal level, an interface can invoke data type transformation passed through *in* parameters to *out* ones and back.

Classes of objects

Objects in the OOM are grouped according to their generic descriptions. That is, OOM bears the following representation [5]:

$M = (Oclass, GK)$,

Where Class = $\{Coloss^i\}$ is a set of classes of object functions or methods with common properties; *GK* is an object graph that reflects relationships between classes and their instances.

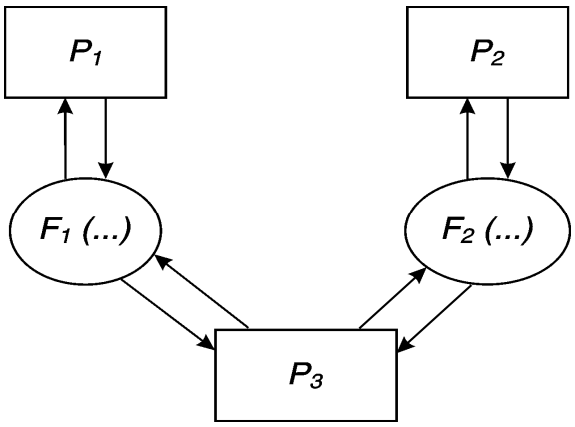

Fig. 2 – Chart of function calls

```
interface F1
{
  void f(in float S [l]);
}
interface F2
{
  const long l=3;
}
Interface P3: F1 ( ), F2 ( )
```

Fig. 3 – Schema description in IDL

Each class is represented as

$$Oclass^i = (ClassName^i, Meth^i, Field^i\} \quad (5)$$

Where $ClassName^i$ is the name of the class; $Meth^i = \{Meth_j^i\}$ is a set of its methods; $Field^i = \{Field_n^i\}$ is a set of variables that determine states of class instances.

Let $Pfield^i \subset Field^i$ be a set of external (public) variables. Each $Pfield_n^i \in Pfield_i$ corresponds to methods *get* <$Pfield_n^i$> and *set*<*Pfieldni*> for setting and selecting the values of the appropriate variables as attributes of objects and interfaces in the OOM and component models.

The set of methods is described as

$Imeth^i = Meth^i \cup \{get<Pfield_n^i>\} \cup \{set <Pfield_n^i>\}$.

It corresponds to an interface $Ifunc^i$ consisting of methods included in $Imeth^i$.

So client class contains techniques for interface-program communication from parameters *in*, *out, inout*, on which are generated for data transfer to the server as instances of the class

Module item (A, B, C)

```
Cost long l=2
Interface A {
Void f (in float s ;}
Interface B {
Cost long l=3}
Interface C: B, A { }
```

Like server class generates an instance of the class for the reversed data transfer to the client.

Description of a program for ClientInterface and ServerInterface co-operation is contained within the instrumental complex ITC http://sestudy.edu-ua.net [13].

3. Modern tools for object implementation

- Component Object Model (COM) for processing components in Basic, C++, .NET, etc.
- Java Beans, Oracle PL-independent standard for component.
- CORBA, consisting of IDL-interface, objects, design of object variants, similar to COM.

- UML, Rational Rose.

Microsoft system uses a subset of data types in all languages for object-oriented modeling in the .NET system, namely: CLS (Common Language Specification), CLR (Common Language Runtime), CTS (Common Type System) system. It includes data type declaration in the CLS specification language for reflecting arbitrary data types to the .NET type system. IBM and CORBA provide software product development from components and services through middleware, which assists in adaptation of system workflow within the environment [2–5, 9].

4. About realization of this approach

Every object will realize OM method or function and is related to other relations, which pass entry and output parameters. Accordant rules of task of objects in CORBA system OM these parameters are described in mediator (stub – for client, skeleton – for server), by the urgent interface. This ORB – standard, it is used now in many modern systems, such as Cloud and Grid The functions of objects will be transformed in the program components and can be used in the component models in the different environments. Objects and components are specified in PL, their passports in the WSDL standard language, and their interfaces in IDL mediator. Presented in standard these elements are configured in the program structure and can be executed in the modern environments. Many aspects of objective realization PS are reflected in ITC web-site http://sestugy.edu-ua.net.

Reference

1. Booch G. at al. Object-oriented analysis and design with applications Addison-Wesley, 2007., – 717 p. 3rd edition.
2. Siegel J. CORBA. Fundamental and Programming, Wesley Co. Publ. Group, John Wiley$Sons.Inc.– USA, 1996.–694 p.
3. Rumbaugh Jet, so on. Object-Oriented Modelling. Standards and Procedures.–Englewood.–Cliffs: Prentice Hall. – 1994.–367p.
4. Robinson K., Beresford G .Object-Oriented SSADM. – New-York.–Prentice Hall, 1994.–279 c.
5. The RAISE Language Group. the RAISE Specification Language. RCS Practitioner Series.–Prentice Hall, 1984.–493 p.
6. Agafonov V.N. Specification of the programs: notion facilities and them organization- Novosibirsk: Science, 1987.-290 p..
7. Lavrischeva E.M., Grischenko V.N. Assembling programming. Fundamental industry of program systems. – Nauk dumka, 2009.–p.371.
8. Grischenko V.N. Method of Object-component development programs Systems. – Problems Programming, 1997, N2.– p. 113–125 (in Ukraine).
9. Lavrischeva E.M. Software Engineering Computer Systems. Paradigms, Technologies, CASE-tools Programming. – K. Nauk. dumka, 2014 – 284 p. (in Russia).
10. Lavrischeva E.M. Methods programming. Theory, Engineering, Practice, K.: Nauk dumka. 2006.– p.451.
11. Object-Component Development of Application and Systems. Theory and Practice, Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk//USA Journal of Software Engineering and Applications, 2014, 7, Published August 2014 in Sires http://www.scirp.org/journal/jsea
12. Lavrischeva E.: Formal Fundamentals of Component Interoperability in Programming.- Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652. Springer, Heidelberg (2010), http://link.springer.com/article/10.1007%2Fs10559–010-9240-z
13. Lavrischeva E.M., Zinkovich V.M., Kolesnyk A.L. and so on. Instrumental-technological complex for Design and training methods of production software systems. Gosluchba Intel. Product of Ukraine. – Registry №45292 of 27.08.2012. –108 p. (in Ukraine).