International Journal of Applied Engineering Research ISSN 0973-4562 Volume 10, Number 24 (2015) pp 45559-45566

Implementation Patterns of Object Static Models for Database Applications: Classical ORM-Patterns and Object-Attribute Approach

Pavel P. Oleynik
PhD, System Architect Software, Aston OJSC, Associate Professor,
Shakhty Institute (branch) of Platov South Russian State Polytechnic University (NPI), Russia, Rostov-on-Don
E-mail: xsl@list.ru

Sergey M. Salibekyan
PhD, Assistant professor, National Research University Higher School of Economics (HSE), Russia, Moscow
E-mail: ssalibekyan@hse.ru

Abstract
This article presents two different approaches to the representation of static models in the implementation of database applications. In order to compare solutions based on selected criteria of optimality the unified model for testing of tools development of object-oriented applications presented in the form of a class diagram language UML is created. As one of the implementations the classical object-relational mapping patterns proposing static models in the environment of a relational DBMS is concerned. An alternative solution widely used by one of the authors is the object-attributive approach. The paper presents the main components of this approach and example of the unified testing model. This paper is the result of years of research and is based on numerous articles published in the Proceedings of the International Scientific and Practical Conference "Object Systems" (objectsystems.ru).

Keywords: Database, Object Models, UML, Object-Relational Mapping Patterns, Object-Attribute Approach

Introduction

At the moment, there are many tools that provide object approach to application development. This is due to the fact that the object-oriented paradigm is dominant in the development of new applications for any subject domains. Object-oriented approach is increasingly used in the implementation of database applications. It is the despite about their strengths and weaknesses of that means. Their main goal is to provide developers all the benefits of object-oriented paradigm for implementation of database applications.

This article presents two different approaches used by the authors in the development of various database. This paper is the result of years of research and is based on numerous articles published in the Proceedings of the International Scientific and Practical Conference "Object Systems" (objectsystems.ru). The described approach is demonstrated by the example of the implementation of unified model for testing of the object-oriented application development tools. It allows reader to draw conclusions about the differences in implemented approaches.

The structure of the paper is as follows. Section 1 describes the using of classical object-relational mapping patterns for the implementation of the static object models. There is the classic patterns of object-relational mapping is discussed in detail in Section 1.1. The section highlights the advantages and disadvantages of each pattern. Section 1.2 describes the implementation of a unified testing model with using of classical object-relational mapping patterns. Section 2 deals with the application of object-attribute approach implementing static object models. Section 2.1 describes the basic elements of object-attribute approach. The section 2.2 highlights the implementation of the unified testing model with object-attribute approach. In last section, it is the conclusions of the work and suggestions for further development is made.

The Use of Classical Object-Relational Mapping Patterns in the Implementation of the Static Object Models

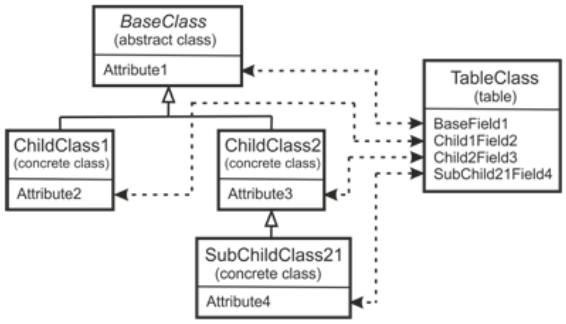
The Classical Object-Relational Mapping Patterns

The practical implementation of the model presented in this section of the text is demonstrated by the using of classical object-relational mapping patterns (ORM). Thus, the object model is mapped into a relational database environment as a set of related tables. This approach is most justified, because the RDBMS is the most popular type of database management systems. Unified environment of rapid development of corporate information systems based on meta model described in [1-2] is used to implement this model. This development environment called SharpArchitect RAD Studio uses a relational DBMS as a repository of information. Because information system is designed in terms of object-oriented paradigm and is implemented in a relational database environment, there is a so-called "object-relational impedance mismatch". ORM-patterns are used to overcome the consequences of the mismatches. A patterns representing the class hierarchy are used patterns are used very frequently.

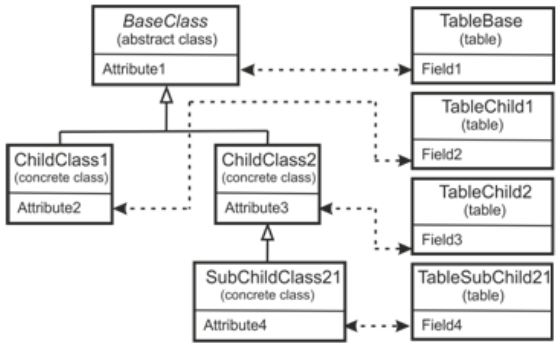
SharpArchitect RAD Studio uses three classic patterns for implementation of object-oriented inheritance relationships of classes in a relational structure tables, presented in Fig. 1 [1].

Consider the basic patterns is presented in more detail. Single Table Inheritance pattern physically represents an inheritance hierarchy of classes in a single relational database table whose columns corresponds to the attributes of all classes within the hierarchy and allows one to display the structure of inheritance and to minimize the number of joins that must be performed to extract information. In this pattern each instance

belonging to the class presents one row of the table. When you create the object values are entered only to a table columns that matches the attributes of the class, and all the others are empty (have a null-value).



Single Table Inheritance pattern



Class Table Inheritance

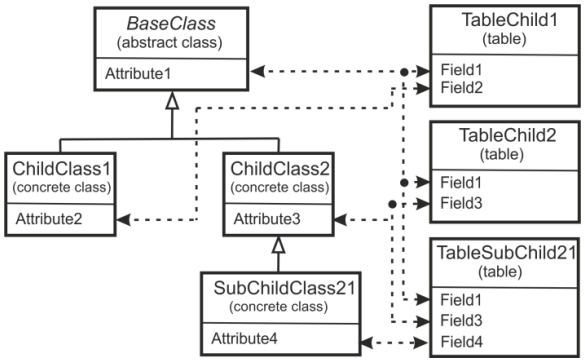


Figure 1. Classical Object-Relational Mapping Patterns which is Used to Represent the Class Inheritance in the Form of a Relational Structure (Relational Tables)

The pattern has advantages:

- The structure of the database contains only one table which represents all classes of whole hierarchy.
- Selection of instances of classes hierarchy do not require joining of tables.
- Moving of fields from a base class to a derived class (as well from the derivative class to the base class) does not require changing of tables structure.

The pattern has disadvantages:

- The structure of the database tables can cause problems, because not all of the columns in the table are intended to describe each domain class. It complicates the process of the system refining in the future.
- If you have a deep inheritance hierarchy with a large number of attributes, many columns can have empty values (null-values). It leads to inefficient use of the available memory space in the database. However, modern DBMS can compress strings containing a large number of null-values.
- Table may be too large and contain a huge number of columns. The main way to optimize the query (to reduce the execution time) is creation of a covering index. However, the index set and a large number of queries to a single table can lead to frequent blockages that causes a negative impact on the performance of software applications.

An alternative pattern is the pattern called Class Table Inheritance representing a hierarchy of classes for one table for each class (as an abstract and concrete). Class attributes are directly mapped to the columns of the corresponding table. Joining of the respective rows of several database tables is main task of the method.

The pattern has the following advantages:

- Each table contains fields corresponding to attribute of a certain class. Therefore a tables are easy to understand and take up little memory space on a hard drive.
- The relationship between the object model and relational database schema is simple and clear.

However, there are disadvantages:

- When you create an instance of a particular class you must to upload data in several tables. It requires natural joining of the tables or a plurality of database calls with followed joining of results in memory.
- Moving of the fields in the derived class or base class requires changing in the structure of several relational tables.
- Superclass table can become weaknesses of performance, since access to such tables will be carried out too often, it will lead to a variety of locks.
- High degree of normalization can be an obstacle for the implementation of unplanned advance queries.

Concrete Table Inheritance pattern presents an inheritance hierarchy of classes using one table for each concrete (non-abstract) class of the hierarchy. From a practical point of view, this pattern assumes that each instance of the class (object) locating in memory is located on a separate row in the table. In addition, each table contains columns corresponding to attributes as a particular class and all of its ancestors.

The advantages are:

- Each table do not contains unnecessary fields, so it is convenient to use in other applications that do not use object-relational mapping tools.
- In the case of creating of objects of a certain class in the application memory and retrieving data from a

relational database data it is selection from a single table, i.e. it is not required to perform relational joins.
- Access to a table is carried out only in the case of access to a particular class, it allow one to reduce the number of locks inflicted on the table and to spread the load on the system.

There are disadvantages:
- Primary keys can be inconvenient for controlling.
- There is no ability to simulate relationships between abstract classes.
- If the class attributes are moved between base classes and derived classes, changing of the structure of several tables is needed. These changes are not as often as in the case of Class Table Inheritance pattern, but they can not be ignored (as opposed to Single Table Inheritance pattern in which these changes are absent).
- If in base class the definition of at least one attribute is changed (for example, change the data type), it requires changing of the structure of each table representing a derived class because a superclass fields are duplicated in all tables of its derived classes.
- Implementation of the method of searching for data in the abstract class require viewing all a tables representing an instance of the derived classes. This requires a large number of database calls.

Selection of a required ORM-pattern depends on the initial logical model, i.e. from the class hierarchy of the domain. At the same time can be used two or more ORM-patterns, as optimization of the structure of a relational database and reducing the number of tables is required. It increases the speed of data retrieval.
After describing SharpArchitect RAD Studio object-relational mapping patterns which are available to the developer we can describe the implementation of the unified testing model.

Implementation of the Unified Testing Model Using Classical Object-Relational Mapping Patterns
In order to simplify the implementation, the three existing class hierarchies is separated in accordance with existing ORM-patterns. The result is shown on Fig. 2.

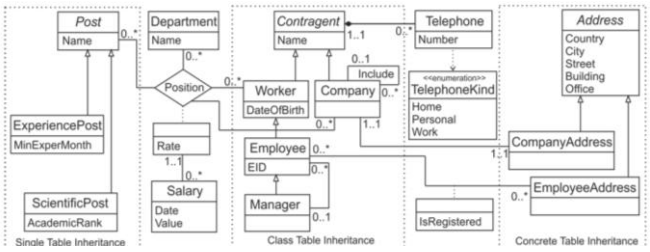


Figure 2. The Use of the Classical ORM-patterns for the Implementation of the Unified Model for Testing Object-Oriented Applications Development Tools

The Single Table Inheritance is used for the class hierarchy Post, ExperiencePost (ScientificPost). As a result, it is assumed that in the RDB one single table (relational table), which will be content instances of all listed non-abstract classes, will be created. Class Table Inheritance pattern is used for the class hierarchy with classes Contragent, Worker (Company), Employee, Manager. I.e. in RDB a separate table is created for abstract or concrete classes. Address class is abstract and has no association with other classes in the model, so a separate table will not be created in the RDB. Two tables (one for each heir) will be created for child classes. I.e. Concrete Table Inheritance was used to hierarchy Address, CompanyAddress (EmployeeAddress). A separate relation table will be created for other classes witch are outside of the described hierarchy.
One of the main features of SharpArchitect RAD Studio is support of multiple inheritance implementing by means of C# language construction interfaces. Used C# language does not support association syntax construction. To represent the binary associations, regardless of the multiplicity, properties (property construction), containing a single value or collection of values is used.
Multiple n-ary association are represented by a separate class. The attributes of these associations (as well as the attributes of binary associations) are converted into property of classes. To simplify data mining all associations are bidirectional, i.e. there are properties both ends of the relevant classes whose type corresponds to the opposite end of the class association. All of the above arguments are presented graphically in Fig. 3.

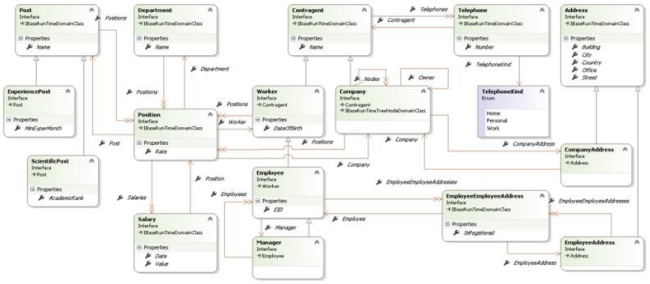


Figure 3. Unified Model for Testing Object-Oriented Application Development Tools, implemented in SharpArchitect RAD Studio in C#

The interfaces language C# are used, so it is impossible to mark abstract classes with italics. Bidirectional associations are shown with corresponding arrows connecting classes. It is used the following approach to implement association. From the "one" side it is declared a property, which have a list type (C# type IList<>) containing the elements, which have a class type locating on the side "to-many". It is declared property in the class of the "to-many" side, whose type is a class locating on the side "one". Association of the "many-to-many" (without attributes) can be represented by two lists declared in opposite classes. The SharpArchitect RAD Studio development environment has a number of base classes that implement the most common functionality. For example, the class IBaseRunTimeDomainClass is the root of all domain classes. To implement the tree structure it is enough inheriting from IBaseRunTimeTreeNodeDomainClass. At the time of code generation it is automatically generating of additional

attributes Nodes and Owner allowing one to save a reference to the parent and subnodes respectively. It is a way to realize recursive association. Syntax construction "enum" is used for ıımplementation of the enumerations and sets.
After applying of the classical ORM-patterns relational database schema of the unified model is obtained. Fig. 4 is depicts the result.

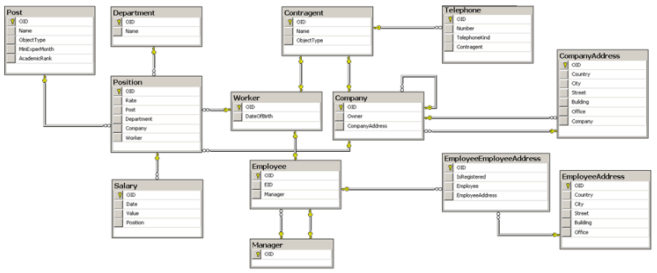


Figure 4. A relational Database Schema of the Implementation of the Unified Model Testing in SharpArchitect RAD Studio

Figure requires the explanation. one single table Post is created for all posts presented by three classes Post, ExperiencePost and ScientificPost. Each table has all attributes of classes. Additionally, there is a column in the table OID, representing an object identifier (primary key in a relational model).
ObjectType column contains the identifier of the class whose objects are stored in the form of table rows. This value is used by the application to create a class of object-oriented programming language and to load the attribute values.
Implementing of Class Table Inheritance pattern causes creation of the table Contragent for abstract class and table Worker, Company, Employee, Manager for the concrete classes. Instances of classes are physically stored in multiple database tables. A copy of the Manager class is stored in all tables.
Implementation of the Concrete Table Inheritance pattern is applicable for classes Address, CompanyAddress and EmployeeAddress. It cause a creation of two tables: CompanyAddress and EmployeeAddress, because CompanyAddress class is abstract. All abstract class attributes is physically stored in tables of specific classes. Association Position have a separate table as well as for the binary association linking the Employee and EmployeeAddress classes. The table EmployeeEmployeeAddress containing foreign keys is created for the binary association.
Note that separate table is not created for the Telephone-Kind enumeration. It is used an approach allowing enumeration values representations as a bit mask and its storing in the form of an integer value, where appropriate attributes are used. So the table has a column Telephone TelephoneKind, SQL-type is Integer.
After analyzing of the above text we can argue that implementation created in a development environment SharpArchitect RAD Studio (Fig. 4) is fully consistent with the unified model for testing object-oriented application development tools.

Use of the object-attribute approach to realize the static object-oriented models[1]

Basic elements of the object-attribute approach

The relational databases (DB) have been used most widely during the last 30 years [3]. Despite this fact, the relational model ceased to satisfy the current requirements, because its restricted abilities to describe the ontology do not allow one to use this model in artificial intellectual systems and the scaling difficulties make the realization of such DBMS in parallel and distributed computational systems more complicated. Numerous ways out of this DBMS crisis [4] were proposed, namely, the object-oriented (OO) approach to DB, the tree-structured DB, the network model, and NoSQL motion (Amazon, Google). But none of them, except for the OO-model which also has numerous drawbacks [5], could sufficiently seriously compete with the relational model.
In this section, we present a new methodic for constructing DB which is based on the object-attribute (OA) approach to organization of the computational process and data structures [6-7]. The OA-DBs belong to the class of network (graph) DBs, i.e., DB is a graph (or an OA-graph) whose vertices are descriptions of objects and semantic connections (relations) and whose arcs are the associations connecting them. To illustrate such a method for the DB organization, we show how to realize the main types of connections between the essences of the OO- and relational DB models and consider an example of a small DB realization on the basis.
This analysis is based on the UML-diagram of classes [8] which is used to define the main essences of the OO-model: class, class attribute, class operation, as well as the following types of relationship: dependence, generalization, multiple inheritance, association, aggregation, composition. The notions of relation multiplicity and role should also be remembered (Fig. 5). Now we describe the realization methods for each type of relations in the OA-DB. We use an OA-language, i.e., a special language for describing the OA-graphs [6-7].
We first describe the class in the OA-DB. So the class (the pattern used to create an object) is a set of fields and methods. In an OA-system, an object is represented as an information capsule (IC); each field of the object is associated with an information pair (IP) contained in this IC. The names of attributes (the attribute index is associated with a unique mnemonics) are the IP attributes, and the class field value is placed in the IP load. The methods in an object are references to a subprogram. The program in an OA-system is a sequence of IP (millicommands). This sequence can be transferred to functional units (FU) which perform computations, data transformations, and their input and output under the control of a millicommands flow.
Let us describe the class in Fig. 6(a) in an OA-language: Man{**Sex**=M **DateOfBirth**=07.02.1970 **FamilyName**=IvanovIvan **OutAge**=YieldAgeProg **SaveCurrentIncome**=SaveCurrentIncomeProg **OutCommonIncome**=OutCommonIncomeProg}, where the

[1] Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged

loads of the last three IP contain pointers to the IC of the subprograms.

The generalization relationship (subclasses) are implemented by adding an IP to the IC of the object description. The attribute of the added IP identifies a subclass, and the load contains a pointer of the OA-graph describing the subclass. For example, the association in Fig. 6(b) is defined as follows: ManFromUniversity{**Student**=StudentPointer{…} **Teacher**=TeacherPointer{…}}, where the dots are replaced by the description of the object corresponding to the Student and Teacher subclasses; Student and Teacher are mnemonics of the attributes. The StudentPointer and TeacherPointer mnemonics are pointers to the capsules containing the respective descriptions of the student and teacher.

The inheritance in the OA-DB is implemented by copying the OA-graph of the parent-object description and adding one or several IP describing the new fields and methods to the root IC. For example, Fig. 6(b) presents the multiple inheritance as follows: the Student-Teacher class inherits the properties of the Student and Teacher classes. In an OA-language, this inheritance is described as follows: StudentTeacher {StudentPointer, TeacherPointer}. If the IC description in an OA-language (but not the IP load) contains the mnemonics of the IC pointer, then this IC (and the entire OA-graph where it is contained) is copied into the formed IC. Thus, all IPs from the Student and Teacher capsules will be copied into the IC, and they will be named StudentTeacher.

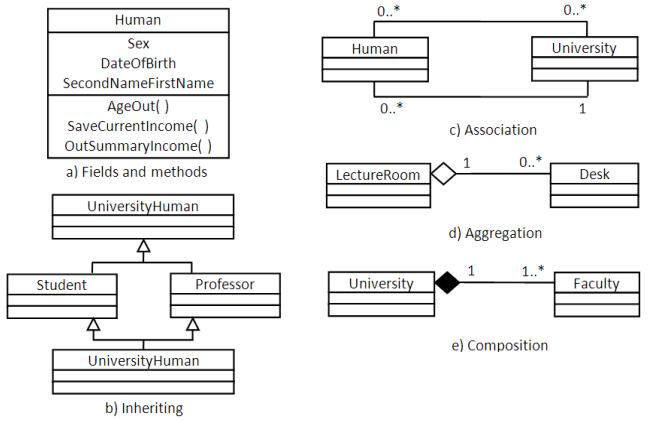


Figure 5 – Basic types of connections in the OO-model of DB

The association relationship are determined by an IP added to the IC of the object description as follows: the IP load contains a pointer to the IC with the description of the associated object, and the attribute contains the role mnemonics. In the OA-approach, the relationship is uniquely determined by two IPs referring to the ICs of the object descriptions. For example, between objects of the class Man and the class University (Fig. 6(c)), there is a relationship where the Man plays the role of a Worker and the University plays the role of an Employer. In an OA-language, the University is described as follows: Man{ … **Student**=University **Worker**=University… } University{ … **Employer**=Man **Teacher**=Man … } Man={**Student**=University **Worker**=University}. A man can work at several universities, and a university can employ several men (a relationship of n-th multiplicity). Then the OA-graph looks as follows: Man1{… **Worker**=University1 **Worker**=University2 …} Man2{… **Worker**=University1 **Worker**=University2 …} University1{… **Employs**=Man1 **Employs**=Man2 …} University2{… **Employs** = Man1 **Employs** =Man2 …}. One can see that in this case, the role name acts as the attribute name.

The composition (an integral part of an object) can, for example, be implemented as follows: to compose a list of attributes whose loads contain the object pointer inseparable from its parent. In this case, if IPs with such attributes are contained in the deleted IC, then the ICs whose pointers are stored in the load are also deleted.

Complex relationship and relationship-classes can be implemented by using IC. For example, it is required to describe the «action» relationship (such a problem arises when a DB is composed starting from meaning of a natural language text) which contains sufficiently many objects: subject (who acts), object (to whom the action is directed), mediator (for example, a tool used to realize the action), etc. Such a relation can look as follows: Action{**Subject**={...} **Object**={...} **Moderator**={...} A**ddressee**={...} **CourseOfAction**={...}...}.

We note that the OA-approach to the DB construction implements all types of connection which can exist in the OO- and relational approaches, and therefore, it can be used to realize applications in any application area.

Realization of a unified testing model by an object-attribute approach

To illustrate the proposed principle, we realize the DB structure of a unified testing model, which is described by using the UML-notation (class diagrams) represented in Section 1. In an OA-language (where // are the comment delimiters), Fig. 6 presents a description of DB whose model is Unified Testing Model. The OA-language complier is contained in the OA-programming and simulation environment, which allows one to simulate the computational process in the OA-system.

```
NewFU={Mnemo="DB" FUType=FUGraph}
// Initialization of atrributes
Department Post
ExperiencePost MinExperMonth
ScientificRank AcademicRank
Position Rate Salary Date
Value Worker DateOfBirth
Company Contragent Employee
EID Manager Include Telephone
Number TelephoneKind
Home#1 Personal#2 Work#3
Name Adress Country
City Street Building
Office CompanyAdress
EmployeeAdress IsRegistred

DB.Set=
>Position1{ // Position1 - IC mark
Post={Name="Full professor"
ExperiencePost={MinExperMonth=12}}
```

```
Department={
Name="Computer science"}
Department={
Name="Applied mathematics"}
Salary={Date="1.10.2014"
Value=4000}
Worker={
DateOfBirth="13.04.1968"
Telephone={Number=1234567
TelephoneKind=Personal}
Telephone={Number=7654321
TelephoneKind=Home}
}
Employee=Employee1{// Mark
EID=12876 Manager=null
EmployeeAdress=
{IsRegistred=true
Adress={Country="USA"
City="New York" Street="Street 20"
Building=10 Office=1}}
Company=
>{Name="IBM"
Telephone={Number=1111111
TelephoneKind=Personal}
Telephone={Number=2222222
TelephoneKind=Work}
CompanyAdress={Country="USA"
City="New York" Street="Street 10"
Building=10 Office=1}
}}}
>Position2{…// Second DB record
```

Figure 6: Listing of the OA-DB whose Unified Testing model.

An OA-program of OA-DB initialization is shown in the listing in Fig. 6. Let us consider it in more detail. A functional unit (FU) processing the OA-graph is constructed in the first line of the OA-program. Further, the attribute and constant mnemonics are initialized (the symbol «#» denotes initialization of a constant, and an isolated mnemonics denotes the attribute initialization). The millicommand (an IP addressed to the FU) with mnemonics «DB.Set» determines the operation «put the pointer to the OA-graph» for the FU «DB», and the description of the OA-graph in an OA-language follows the symbol «=». The symbol «>» denotes the beginning of a new record in the list (the list is organized from the essences «Position», and this list «sews» all records together in a unified DB). After the initialization by the FU «DB», one can modify the OA-graph and seek the information in it. The OA-graph created in the OA-programming and simulation invironment is illustrated in Fig. 8 (a special program component was realized to derive an OA-graph in the OA-programming and simulation environment). The indices of the information capsules (IC) are given in curly brackets in Fig. 7, i.e., since the OA-graph is a graph of any arbitrary topology but the screen shows only its frame, the indices allows one to trace all connections between the vertices which do not enter the frame; in this case, after the symbol «=» (notation of the IP load), the index of the IP referring to the load is placed (for example, «ID_14»).

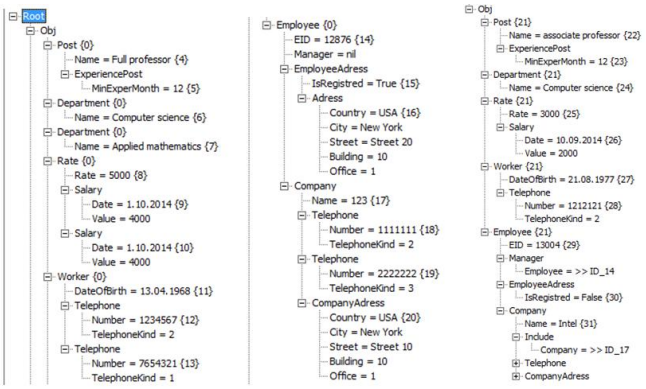


Figure 7: DB representation as an OA-graph

The OA-approach ensures the DB key properties such as description of any ontology, computational scalability, DB integrity, ergonomicity (DB queries are entered with the natural language) [9], and a high speed of search in DB (a special search technique based on the use of the IC indexation was developed for the OA-DB).

The OA-DBs belong to the class of network (graph) DB, where the ontology description is a set of vertices denoting objects and arcs which determine relationship between the objects. The network DB has no restrictions on the topology, and therefore it can potentially describe any ontology. This is its qualitative distinction from the relational and OO-models: in the first case, it is practically impossible to describe the ontology with a large number of connection types (it is necessary to create its own relation (table) for each type, which is very cumbersome), and in the second case, there is a restriction on the topology (a «tree»-type graph).

The OA-DB scalability is attained because the computations are controlled by using the dataflow [10]. The dataflow allows one to «untie» the data addressing from the computer common memory. Namely, the data are accessed through functional units (FU) which are virtual, i.e., they are not rigidly depended with the hardware of computational nodes. The set of FUs forms the so-called OA-image, i.e., a virtual computational system. Then the OA-image is «imposed» on a specific computational system (CS) by distributing the FUs over the CS computational nodes and by adjusting the FUs themselves and the routers responsible for the transfer of IPs between the computational nodes. To change the CS configuration, it is only necessary to redistribute the FUs between the computational nodes and to introduce new adjustments of the routers [6].

The OA-DB wholeness is ensured by the format and technic of the OA-graph processing. So the OA-DB modification consists in addition or removal of several IP from the IC composing the OA-graph. We assume (Fig. 8) that there is an IP with attribute «1» in the IC, and then we add an IP with attribute «2» to the same IC. Now we can seek the data according to either attribute «1» or attribute «2» independently, i.e., if we seek the data with attribute «1», then we ignore attribute «2», and conversely. Thus, we can distinguish several subgraphs by analyzing the IP attributes

only in one OA-graph… Namely, we can change the DB structure without fear for its wholeness, and therefore we need not perform a very laborious refactoring, which cannot practically be avoided in any serious modification of big OO-DB.

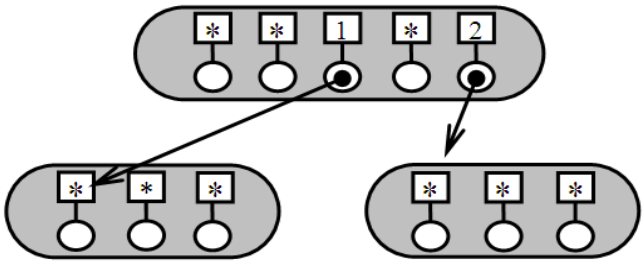


Figure 8: Assurance of OA-DB integrity in its modification

The ergonomicity is ensured because the queries to the OA-DB are in the natural language, i.e., an query is transformed in its OA-graph, and then it is determined whether the query is a subgraph of the OA-graph of DB. The technic of transformation of a natural language text into an OA-graph is discussed in [9]. The answers to the queries can also be given in the natural language.

Similarly to the OO-approach, the OA-DB gives wide opportunities for the data abstraction. For example, an OA-graph can be constructed according to the «tree» topology. In this case, an IC of a higher level encapsulates the information structure of the lower level (the IC contains the basic characteristics of the lower level). If the OA-graph is of any arbitrary topology, then we can distinguish subgraphs in it (for example, according to the principle of description of any object, set, or phenomenon) and include the IC describing the main characteristics of such subgraphs into the OA-graph. Now, if necessary, the user can operate with this IC without referring to a detailed description of the object.

Because of IC indexation, the search of information in OA-DB is sufficiently fast. As previously noted, the search in the OA-DB is the search of a subgraph (subgraphs) in the OA-graph of DB which coincides with the enquiry-graph [9]. The search process becomes simpler because the OA-graph vertices are, so to say, marked by the IP contained in the IC. Therefore, we can propose the following methodic for seeking subgraphs, which consists of several stages. The first step is to index all vertices of the OA-graph of DB and the enquiry. The second step is to compose the following two lists of the IP attributes: the first list of attributes encountered in the OA-graph of DB and the second list of attributes of the enquiry OA-graph. If some enquiry attributes are not contained in DB, then the search is complete and the user receives the negative result message. Otherwise, all IC containing attributes from the enquiry-graph are distinguished in the DB graph. The third step is to group the IP obtained in the OA-graph according to capsules: if the obtained IP have the same IC index (i.e., if they belong to the same capsule), then these IP are united in a group. The fourth step is to match the groups: if a group does not coincide with any other group in the enquiry-graph, then it is removed. As a result, if all groups are removed or the number of groups becomes less than that in the enquiry-graph, then the search is complete with a negative result. The fifth step is to match the remaining groups with the enquiry-graph IC. The search of the required attributes can be enhanced by using the hasting, which significantly increases the speed of computation. Such an algorithm has time complexity of the order of $N*M$, where N is the number of IP in DB and M is the number of IP in the enquiry-graph.

It should be noted that, theoretically, the OA-DBMS has sufficiently good perspectives, because it preserves many positive properties of DBMS, but it also has the following drawbacks: first, the main memory consumption is excessive (in OA-DB, two IPs are distinguished for each connection, because the OA-graph arcs must be bidirectional); second, because the addition of new data structures does not violate the old data structures, OA-DB may contain unnecessary data structures which are already forgotten by the designers but which occupy the memory and hinder the search of the required information.

Conclusions

The article presents two modern approach to the implementation of the static models, designed in the implementation of object-oriented applications. As an example, for a unified model represented the language class diagram UML. The described solutions are the result of years of work of the authors and tested in applications, many users actually used. The further development of these approaches the authors see the implementation of the principles of formation of the graphical user interface. Also proposed to develop formal methods of description of the presented solutions.

References

[1] 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

[2] 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).

[3] C.J. Date Introduction to Database Systems (8th edition) 2003

[4] Why Programmers Don't Like Relational Databases 2007. URL: http://typicalprogrammer.com/programmers-vs-rdbms/

[5] Gabriel, R. Objects Have Failed: Notes for a Debate. (retrieved 17 May 2009). http://www.dreamsongs.com/Files/ObjectsHaveFailed.pdf

[6] S.M. Salibekyan, P.B. Panfilow Object-attribute

architecture for design and modeling of distribute automation system. // Automation and remote control. Volume 73, 2012, Number 3, 587-595, DOI: 10.1134/S0005117912030174

[7] Salibekyan S. M., Panfilov P. B. Dataflow Computing and Its Impact on Automation Applications // Procedia Engineering. 2014. No. 69. P. 1286-1295. URL: http://www.hse.ru/pubs/share/direct/document/126338249

[8] Martin Fowler, Kendall Scott, (2003). UML Distilled: A Brief Guide to the Standard Object Modeling Language, Third Edition, Addison-Wesley.

[9] Salibekyan S.M., Khal'kina S.B., Tinovitskiy K.D. Object-attribute Approach for Natural Language Processing // 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. 80-86 (In Russian), http://objectsystems.ru/files/Object_Systems_2014_Proceedings.pdf

[10] Jurij Silk, Borut Robic and Theo Ungerer «Asynchrony in parallel computing: From dataflow to multithreading» Institut Jozef Stefan, Technical Report CDS-97-4, September 1997.

[11] Pavel P. Oleynik, Nikolay V. Kuznetsov, Edward G. Galiaskarov, Ksenia O. Kozlova. Domain-Driven Design of Information System for Queuing System in Terms of Unified Metamodel of Object System. International Journal of Applied Engineering Research, ISSN 0973-4562, Volume 10, Number 15 (2015), pp. 35229-35238.

[12] Pavel P. Oleynik, Nikolay V. Kuznetsov, Edward G. Galiaskarov, Natalia E. Borodina. Model-Driven Design and Implementation of Scientific Data Management Information System. International Journal of Applied Engineering Research, ISSN 0973-4562, Volume 10, Number 15 (2015), pp. 35239-35246.