Keywords
Multiple Inheritance, Design of Information Systems, Database, UML

1. Introduction

Information systems are the integral element of the modern enterprise now. They provide collecting, storage and information processing in the course of the solution of problems of any applied subject domain. Information systems are widely applied in work of various organizations. They won the popularity thanks to a set of pluses from which it is possible to distinguish the following:

1) IS increase productivity of the enterprises;
2) IS can increase quality of the rendered services, also improve quality of the made goods;
3) Considerably to reduce time of some processes of production by their automation.

Application of IS is most popular in services industry since they allow to facilitate work the staff of the organization and to increase quality of customer service. That in turn leads to improvement of many indicators of work of the organization in general and to profitability increase in particular. Therefore the management of such enterprises notes the
2. Using Multiple Inheritance

Administrators of modern salons don't use now the handle and a sheet of paper to record clients on services, and do it by means of special developed programs in which they write down new clients, note records on services, conduct client base, trace work of each master, the goods remains in a warehouse and conduct settlement operations with each client.

At design of information system always special attention is paid to the development of structure of a database. Thus we allocate three main design stages: conceptual, logical and physical. Conceptual design of a database allows to describe high-level model and begins with creation of the conceptual model of data of the enterprise completely independent of any details of realization. The chosen DBMS type, structure of programs of the appendix, the used programming language, a certain hardware platform, questions of productivity and any other physical features of realization concern to the last one.

The following stage is logical design of a DB which is understood as designing of information model of the enterprise on the basis of the existing certain models of data, but without specifics of the used DBMS and other conditions of realization [1].

Generally, logical design of a database consists in transformation of conceptual model of data to logical model of data of the enterprise taking into account the chosen DBMS type (in our case, in concepts of object DBMS). The logical model of data is information source for a stage of physical design. It provides to the developer of physical model of data by means of carrying out the comprehensive analysis of various aspects of work with data that is important for a choice of the effective design decision.

The last stage is physical design of a database which is understood as the description of certain realization of the database placed in external memory. The physical project describes the basic relations, defines the organization of files and structure of the indexes applied to ensuring effective access to data and also regulates all corresponding restrictions of integrity and a measure of protection [1].

Physical design of a database provides acceptance by the developer of a final decision on ways of realization of the created base. Therefore physical design is surely made taking into account all features of the used DBMS.

Fig 1. Fragment of the chart of classes.

The modern languages of programming are object-oriented and operate with such key concepts as encapsulation, inheritance and polymorphism. Thanks to the existence of accurately formulated principles of development all new software products are developed in OO-languages.

Design of object-oriented appendices is carried out by means of the unified language of modeling of UML [2] which allows to describe not only static, but also dynamic aspects of system. In figure 1 the fragment of the chart of classes represented in the notation of UML is presented.

The presented fragment of hierarchy of classes is a kernel for the information system of beauty shop developed by authors in the given article. The document Order represents preliminary record on a session. At the record the administrator of salon creates a set

We will note that many modern OO-programming languages, such as C# and Java don't support multiple inheritance of classes in pure form. However in many languages there is such syntactic design as interface which submits only the description of declarations (without providing realization) and allows to simulate multiple inheritance. This described design of the C# language was
used at realization of a fragment of the chart of classes presented in figure 1.

Thus the classes Product and Service are inherited from two basic abstract classes (Commodity and IBase Run Time Tree Node Domain Class). Root IBase Run Time Tree Node Domain Class is the built-in system class provided by the environment of development allowing to organize hierarchy of copies of certain classes. Treelike structures are very convenient for grouping services and goods in category. Thus the level of hierarchy of the described classes has no restrictions on depth. The current realization won't allow to mix various knots in uniform hierarchy. I.e. it is impossible to mix services and goods in one tree. It would be difficult to be reached at single inheritance (at inheritance of Commodity from IBase Run Time Tree Node Domain Class).

The basic class Commodity is abstract and represents object of consumption. Its introduction to hierarchy is caused by need of the organization of association with Order Row which represents a line of the demand and contains the reference to goods or to service.

Despite the existence of a number of the key advantages allowing to present adequate realities of the world in the software form by means of multiple inheritance it has a number of shortcomings [3-4]. Most often note coincidence of names of variables and methods at ancestors of a class and ambiguity of a way of inheritance in a case, more than two-level hierarchy. For a solution the copy of a class needs to be given to the demanded basic class. We will note that all shortcomings listed in various sources don't cause essential damage, but give a number of the advantages described earlier. These advantages allow to avoid errors of input of information and to carry out primary validation that is especially actual in large information systems.

3. Object System Metamodel Structure

For implementation we used object system metamodel which describes in [5-11]. The metamodel has the advantages:
1. Classes added that represent attributes of different data types such as geometric objects (points, lines and closed figures), geographical coordinates and attributes that allow representing instances of classes.
2. Calculated classes added in the system, instances of which are not stored in the database and used for the implementation of algorithms and business processes of application domain.
3. Designed and implemented class hierarchy representing default values for the class attributes. In this case, it is possible to inherit and override values in derived classes (polymorphism).
4. Added various options (in the form of a sets/enums) for classes that allow controlling behavior of domain class instances. For example, it is possible to specify that the class implements the Singleton design pattern and thus allows restricting the ability to create more than one instance, and the ability to delete.

For realization of the described information system our own environment which was tested on information system for fast food restaurants presented in work [12-14]. To implement the unified rapid development of enterprise information systems, Sharp Architect RAD Studio was used. Fig. 2 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. The 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) Not InheritableClass 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 Custom Attributed Class is used to represent metaclasses that have the attributes. Metaclass Domain Class 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 Computational Class<TBase Class> is the base for all calculated metaclasses i.e 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 Domain Class).

Metaclass Method Parameter Class 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 Sharp Architect RAD Studio.

Abstract metaclass Code Computational Class<TBase Class> is the base for calculated metaclasses implemented using code in the language of C#. Query Class 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). Helper Class 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. 2.

Root abstract metaclass representing an attribute is Abstract Attribute. Classes inherited from Virtual Attribute 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. System Attribute allows to describe the attributes that are system and are
presented at the language of C#. Metaclass Generated Attribute 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 Concrete Attribute 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 Typed Attribute<TDefault Value> was added. Type Attribute is used to represent the properties whose values can store a reference to the data type of the C# language.

Metaclass Classed Value Attribute<TValue Class, TDefault Value> is used to represent attributes whose values are the instances of the different instances of entity classes present in the domain. Metaclass Not Inheritable Classed Value Attribute<T Value Class, TDefault Value> retains instances of non-inherited classes. For example, Enum Attribute inherited from the one described is used to store values of enumerations / sets.

As can be seen, Sharp Architect 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 [12-14].

Fig 2. Fragment of a unified metamodel of object system.
4. Conclusions

In the future, it is planned to implement a mechanism for automatic construction of physical model using available instances of classes. The presented solution can be used in designing any database application and metamodel can be used as a basis to create a MDA tool.

References


